8-1-2007

Applicability of clustering to cyber intrusion detection

Gilbert Hendry

Applicability of Clustering to Cyber Intrusion Detection

by

Gilbert R. Hendry

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Supervised by

Assistant Professor, Computer Engineering Dr. Shanchieh Yang
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
August 2007

Approved By:

Dr. Shanchieh Yang
Assistant Professor, Computer Engineering
Primary Adviser

Dr. Juan C. Cockburn
Associate Professor, Computer Engineering

Dr. Michael E. Kuhl
Associate Professor, Industrial Engineering

Thesis Release Permission Form

Rochester Institute of Technology

Kate Gleason College of Engineering

Title: Applicability of Clustering to Cyber Intrusion Detection

I, Gilbert R. Hendry, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or in part.

Gilbert R. Hendry

Date

Acknowledgments

I would like to give most of my thanks to Dr. Yang. It is to him that I owe the completion of my thesis in a semi-timely manner. His advise and insight were invaluable. Also thanks to Brian Argauer, Stephen Byers, and Dan Fava for being in the same situation. Lastly, thanks to CUBRC in Buffalo, NY and Air Force Research Labs in Rome, NY for technical guidance and support.

Abstract

Maintaining cyber security is a complex task, utilizing many levels of network information along with an array of technology. Current practices for combating cyber attacks typically use Intrusion Detection Systems (IDSs) to passively detect and block multi-stage attacks. Because of the speed and force at which a new type of cyber attack can occur, automated detection and response is becoming an apparent necessity. Anomaly-based detection systems, such as statistical-based or clustering algorithms, attempt to address this by analyzing the relative differences in network and host activity. Signature-based IDS systems are typically more accurate for known attacks, but require time and resources for an analyst to update the signature database. This work hypothesizes that the latency from zero-day attack to signature creation can be shortened via anomaly-based algorithms. In particular, the summarizing ability of clustering is leveraged and examined in its applicability of signature creation.

This work first investigates a modified density-based clustering algorithm as an IDS, with its strengths and weaknesses identified. Being able to separate malicious from normal activity, the modified algorithm is then applied in a supervised way to signature creation. Lessons learned from the supervised signature creation are then leveraged for the development of unsupervised real-time signature classification. Automating signature creation and classification via clustering turns out satisfactory but with limitations. Density supports for new signatures via clustering can be diluted and lead to misclassification.

Contents

List of Figures

List of Tables

Glossary

G

ground truth The absolute status, whether malicious or normal, of network activity data. In the KDD data, these labels come as either ‘normal’ or one of 22 attack types., p. 5.

I

IDS Intrusion Detection System, p. 2.

R

ROC Receiver Operating Characteristic, p. 36.

S

signature A model or template used to compare against cyber data to identify malicious activity., p. 4.

similarity measure In clustering algorithms, the metric which determines how far a data point or another cluster is from a particular cluster or cluster centroid., p. 6.

sniffing Copying physical or transmission layer data packets on a network whose destination is not the copier., p. 2.

supervised method Any classification method that uses some model or information from ground truth., p. 5.

T

TCP Transmission Control Protocol, p. 28.

Z

zero-day attacks Attacks that are new to the cyber security community, and therefore have no specific counteracting measures against them., p. 33.

Chapter 1

Introduction

Cyber security today is a problem growing more complex with increasing network sizes, amount of information, and sophistication of attacker methods. One key component of network analysts' tools is an Intrusion Detection System (IDS). These systems attempt to analyze both host and network data to detect the presence of occurring malicious activity.

Two prevailing methods of intrusion detection are anomaly-based and signature-based analyses. Many tools exist using methods from one or both of these categories, and achieve varying results. In general, anomaly-based algorithms may miss a large volume of abnormal activity and also trigger false positive alarms from legitimate yet rare events. However, one key advantage of anomaly-based algorithms is that they often require no *a priori* knowledge and can therefore theoretically adapt to new attack methods. Signature-based algorithms usually have a high detection rate, but take time and resources to update the database of attacks they are reliant on. A plausible approach would be an anomaly-based method that can update or incorporate signatures either created by an analyst or automatically created from another source.

While much work has been devoted to Intrusion Detection Systems, automatic examination of host and network data in real time with high accuracy is still not satisfactory. Many proposed algorithms have not addressed their reliability with varying amounts of malicious activity or their adaptability for real time use. The ability of a system to perform and adapt in real time is necessary because attacks can often be automated, and therefore performed very quickly on a network.

This work will extend an existing unsupervised algorithm to real-time use and investigate its performance improvement by incorporating adaptive signatures.

1.1 Intrusion Detection Systems

Current Intrusion Detection Systems (IDS) technology monitors both network and host activities to detect malicious attacks. Network data is collected by 'sniffing', or copying, transmission layer activity at some critical point in a network. Systems that are solely network based, such as [41], can arguably miss valuable activity occurring internal to a single host. Alternatively, host data can come from a variety of automatically and explicitly collected audit logs, depending on the Operating System, configuration, and normal usage of the host. Because of the variety and volume of this data, it is often unclear as to the best way to use the vast amount of information available on each host.

The methods that IDSs employ can typically be grouped into two categories: signature-based and anomaly-based Sensors. Signature-based analyzers compare events against predefined models looking for either specific kinds of events or specific patterns of events. This method requires one or more experts defining and constantly updating these signatures. Anomaly-based analyzers attempt to identify unusual events or unusual patterns of events, assuming that cyber attacks produce rare or different effects that appear in network or host data. Anomaly detection is focused on attempting to catch new attacks as well as old ones, but it often comes with the cost of being less accurate because of unusual but legitimate user behavior or varying amounts of malicious activity.

1.1.1 Signature-based Systems

Signature-based intrusion detection is the predominant method in commercially available IDS systems, *e.g.*, [41, 31, 18]. In a signature-based system, activity is compared to a variety of specific models that are historically labeled as malicious. A signature-based IDS will most often have a very high detection rate of known malicious activity due to the

predictability of many attacks and the precision of the models that is possible. According to Frederick [13], the following are some commmon examples of signatures with a range of complexity used in typical IDS technology.

- Connection attempt from a reserved IP address. This is easily identified by checking the source address field in an IP header.
- Packet with an illegal TCP flag combination. This can be found by comparing the flags set in a TCP header against known good or bad flag combinations.
- Email containing a particular virus. The IDS can compare the subject of each email to the subject associated with the virus-laden email, or it can look for an attachment with a particular name.
- DNS buffer overflow attempt contained in the payload of a query. By parsing the DNS fields and checking the length of each of them, the IDS can identify an attempt to perform a buffer overflow using a DNS field. A different method would be to look for exploit shellcode sequences in the payload.
- Denial of service attack on a POP3 server caused by issuing the same command thousands of times. One signature for this attack would be to keep track of how many times the command is issued and to alert when that number exceeds a certain threshold.
- File access attack on an FTP server by issuing file and directory commands to it without first logging in. A state-tracking signature could be developed which would monitor FTP traffic for a successful login and would alert if certain commands were issued before the user had authenticated properly.

It is clear from the above examples that signatures usually target very specific features of observed activity. Time and resources are therefore needed for a signature-based IDS to adapt to new attacks. Because cyber attacks can change drastically in short periods of

time, signature databases must be updated by analysts frequently. Also, signature-based IDSs can have varying degrees of sensitivity which may produce varying amounts of false positives. For example, Patton *et al.*[34] identified that Snort can have a high false positive rate. Figure 1.1 shows a system-level diagram of the components of a signature-based IDS. Becasue an analyst is usually involved in this loop, signatures can take a relatively long time to be updated. An extremely accurate IDS can also incorporate an automated response, such as automatic firewall rule creation to block malicious activity, though this technique is difficult to use reliably without creating a denial of service for legitimate activity.

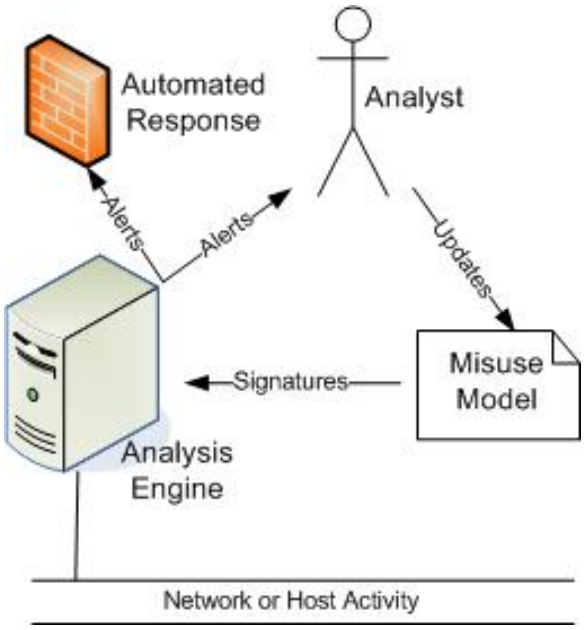

Figure 1.1: Signature-based IDS

Some systems have been proposed which improve on signature-based IDS performance. For instance, Carey *et al.*[5] investigated different methods of pattern matching in a heterogeneous signature-based IDS. Other heterogeneous systems discussed later in Section 1.3 employ signature-based methods to achieve high accuracy. However, because of the precision of signatures which allows them to be highly accurate, new attacks can often go undetected. Though reliable for detecting known attacks, signature-based systems take time to adapt to new attacks and can produce floods of false positives.

1.1.2 Anomaly-based Systems

Anomaly-based detection schemes aim at finding attacks based on their relative differences to normal activity, and typically use statistical profiling, clustering, or other machine learning techniques. Figure 1.2 shows a diagram of a typical 'supervised' anomaly-based system. A system that uses a model created from training data is typically refered to as a supervised method because it requires that the training data be labeled as ground truth by an analyst. Though not all anomaly-based algorithms are inherently supervised, any method which relies on a model which is compared to observed activity must create this model under supervision. Without the components in Figure 1.2 labeled under 'Supervision', an anomaly-based method would rely on the differences in the observed activity alone, and not differences between the observed activity and historical activity.

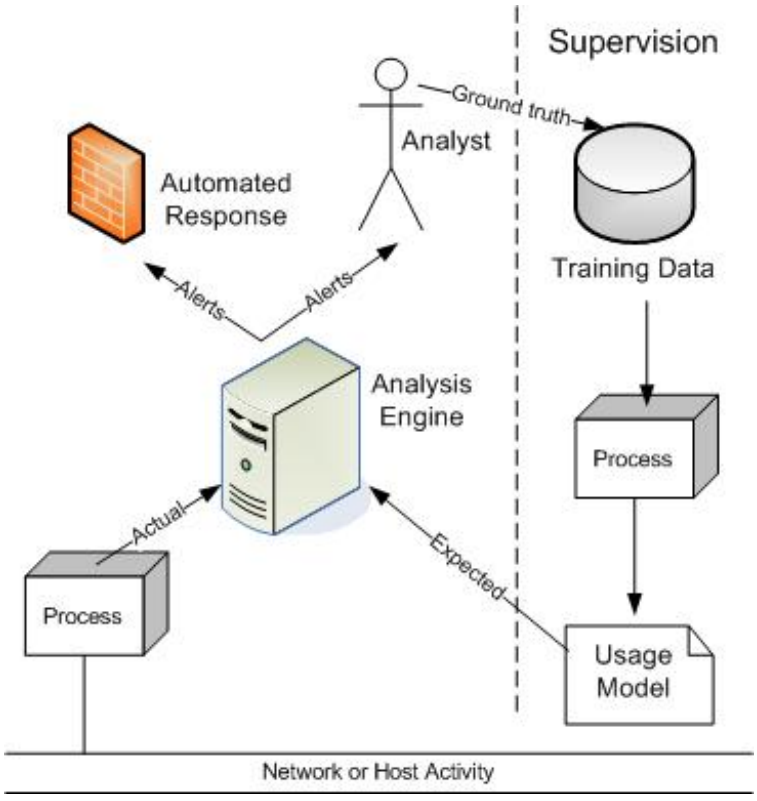

Figure 1.2: Anomaly-based IDS

In a Statistical Profiling analysis, normal user behavior is defined in terms of the statistics for different kinds of events. These statistics are collected during an initial supervised profiling stage. Typically, if the observed statistics deviate from these numbers, then an alert is triggered. Some well-known systems rely on this type of anomaly detection on many levels [18, 35, 38, 23, 24, 21].

There are, however, some inherent problems with statistical profiling. If the data used to train a normal behavior model contains malicious activity, it may become part of the model in such a way that it can never be detected later in real analysis. Statistical profiling is also vulnerable to profile drift. If normal behavior changes over time, as it often will, it may be regarded as malicious because the profile is out of date. Some methods attempt to address this, though it is difficult to gauge how these adjustments would perform in every scenario. Finally, like any anomaly-based system, statistical profiling is vulnerable to varying amounts of malicious activity. A large steady amount of malicious activity could deceive the IDS into eventually labeling it as normal. These problems require more sophisticated methods to reliably detect malicious activity.

Another category of Intrusion Detection methods include machine Learning techniques such as neural networks and Hidden Markov Models (HMMs). Like statistical profiling, these techniques attempt to learn either malicious or normal behavior. Machine Learning, however, learns the characteristics of the behavior, and not just the relative frequency of its occurance. For instance, neural networks describe a classification system as a series of small decisions. This model is trained using ground truth to describe behavior by identifying the series of decisions more frequently made when presented with a type of activity. These paths can then be used as a model for comparison to observed activity.

Another category of anomaly-based intrusion detection is clustering algorithms. These often unsupervised methods attempt to group data based on similarity measures determined over a high dimensionality in hopes of isolating interesting groups of data possibly in subspaces not obvious to an analyst. Like any anomaly-based system, clustering aims at capturing new and evolving attacks as quickly as possible by identifying the relative differences

between normal and malicious data. Clustering has been used in data mining for decades in various domains, though it has only recently been applied to intrusion detection. It has not yet made its way into popular commercial tools probably due to unreliable performance, though clustering is still a topic of IDS research [15, 51, 32, 39, 40, 19, 12, 26].

Because clustering often does not examine temporal relationships between data points, it is not susceptible to any kind of profile drift. However, like many truly anomaly-based detection systems, clustering is often based on the assumption that malicious activity is in fact anomalous. This assumption makes it vulnerable to varying amounts of malicious activity. This characteristic along with its lack of supervision can often make performance heavily reliant on user parameters. In order for clustering to be of practical use in intrusion detection, it must demonstrate that it can be reliable and also be useful for detecting new kinds of attacks. Clustering algorithms are not reliant on *a priori* knowledge, and can usually summarize many high-dimensional data points, which makes them a point of interest in the intrusion detection community.

1.2 Clustering Algorithms

Clustering algoriths by themselves have been used for decades as summarizing tools for large data sets. When applied to anomaly-based IDS systems, they attempt to find anomalies within data by grouping it in such a way that unusual data points are left by themselves or in small clusters. These anomalies are often harder to spot by an analyst manually because the clusters are formed from subspaces within the data set. Oh and Lee [33] point out that because statistical profiling attempts to summarize a user's behavior, it can become very inaccurate when the normal behavior's deviation is large. Clustering provides a major advantage over other algorithms because it identifies both outliers and dense regions in a data space without using *a priori* knowledge. The best clustering algorithm for intrusion detection purposes will therefore use the most amount of information in a data set to distinguish between malicious and normal activity.

Currently, there are many clustering algorithms across many different data mining applications. One species of clustering algorithms uses distance as a similarity measure when forming clusters. Many different kinds of distance formulas that can be used in a classifier are identified by Luke [27], including Euclidian, Manhattan, Chebyschev, Squared Chi-squared, and Canberra distances, as well as formulas based on Cosine Similarities and Pearson's Correlation Coefficient. K-means [28] is one distance-based algorithm that led to the development of many others [15, 32, 48, 50, 16, 36, 52]. Typically, data is grouped around established cluster centroids, the number and location of which may be determined iteratively. Though using this kind of similarity measure is intuitive for data existing within the real numbers, it is not clear how symbolic fields, such as protocol (e.g. 'tcp'), should be used. Since symbolic fields are common in both network and host logs, many distance-based algorithms solve this problem by assigning a constant value to the distance between two symbolic points. Because it is not readily clear what this constant value should be, distance-based clustering algorithms do not effectively use the information contained in these fields.

The opposite species of clustering algorithms is the density-based approach, which includes work such as [1, 11, 30, 14, 46]. A density-based approach requires a user parameter that indicates a density threshold, which specifies how many instances of a value are needed for that value to be considered 'dense'. Clusters are then formed by analyzing the dense regions of a data set. It is less intuitive how to measure the similarity of any two data points in a density-based cluster, though it can be argued that each data point in a cluster is equally far apart. Measuring inter-cluster distances would require another density based analysis of the formed clusters, which can conceivably be useful. Because density-based approaches are dependent on a user parameter and also summarize information in determining dense subspaces in the data, it could be argued that they do not necessarily take advantage of all of the information of every field in the data set. However, density-based algorithms are better suited to using fields with symbolic values.

Clustering algorithms may also exhibit other characteristics, such as their usefulness

for real-time use. Some algorithms [32, 4, 3] address this issue either by treating log data as a data stream or by otherwise analyzing data with a temporal aspect. Other algorithms [51, 40, 12, 26] use other methods such as genetic selection and fuzzy logic to enhance clustering performance.

In general, clustering has been shown in some algorithms to perform well as an IDS, though for small, specially selected data sets. Regardless of the proposed enhancements to Clustering algorithms over the years, no existing methods clearly show their performance for varying amounts of heterogeneous malicious activity. A simple algorithm is needed with intuitive improvements available to show the ability of a clustering algorithm's ability to separate malicious and normal activity, and the significance of the improvements to these algorithms.

1.2.1 Simple Log-file Clustering Tool

The clustering algorithm selected is a simple density-based approach proposed by Risto Vaarandi [46]. In his paper, he refers to the software that implements his algorithm as the Simple Log-file Clustering Tool (SLCT), which we will adopt as the name of the algoritm itself. Vaarandi argues that a density-based approach takes advantage of the structure of log files, where a distance-based algorithm does not. SLCT was designed simply as a log analysis tool to aid a user in summarizing a data set. However, we propose to investigate it's use in a number of ways, including as a stand-alone IDS.

Vaarandi's algorithm has many advantages over other methods for our purposes. One benefit is its simplicity. Implementation of the algorithm is easily constructed, and areas for fitting the algorithm to intrusion detection are apparent, including real-time adaptation. Clusters are also formed in such a way as to maximize their fixed attributes, or in other words, by finding the largest possible subspaces. This means that the most information possible for a density-based approach is used from the data set.

1.3 Combining Signature Analysis with Clustering

Both signature-based and anomaly-based methods have advantages and disadvantages. Signature-based methods, while often more accurate, require time and effort to update their signatures. Anomaly-based methods, though vulnerable to varying amounts and types of normal and malicious activity, can adapt to new attacks because their detection method is based on a relative approach. To take advantage of both types of intrusion detection, one approach is to use reliable signatures created from a supervised clustering algorithm, while updating them based on the results of a similar unsupervised clustering technique. Signatures must be updated in a way that retains the signatures' useful information, while adapting to changing attack methods.

Some existing systems attempt to combine the advantages of both types of systems. One approach is to use a supervised anomaly-based system to create signatures that are later used for detection [49, 25]. Some systems propose combining separate processes of signature- and anomaly- based methods to decrease false positive rates [47, 9]. When trained correctly, these systems can be very accurate. These works do not provide, however, schemes for updating signatures automatically. Other systems propose using artificial intelligence and other machine learning techniques to adapt an intrusion detection model [2, 29, 6]. Though these systems are adaptive, not all of them demonstrated their accuracy with a range of heterogeneous dynamic activity.

In commercial Intrusion Detection Systems, new signatures are often formed by an expert using network or audit data that was determined to be malicious. Automating this process in an unsupervised way therefore requires some reasoning that can reconcile existing signatures with new attack data identified by an anomaly-based system. We define a signature as a format for malicious or normal activity, having one or more fixed attributes. Because of imperfect signatures and changing attack methods however, new information identified as malicious or normal must update, add to, or invalidate existing signatures to reflect current methods.

Though much work has previously been done in the field of intrusion detection, new innovations are needed to keep up with constantly changing attack methods. Many signature- and anomaly-based systems exist using a variety of techniques. Systems that combine the two obtain good performance, but do not automate the adaptation process. The proposed work is different from existing methods in that it retains the integrity of proven signature-based methods, while attempting to automatically adapt to new attack methods, thus attempting to reduce the latency between attack occurrence and signature creation.

1.4 Problem Statement

The work will address specific problems by applying different algorithms to intrusion detection. First, it is necessary to determine if a density-based clustering algorithm such as SLCT is useful as a stand-alone IDS. The goals of many studies mentioned above have been to create a successful Anomaly-based IDS. Many have claimed good performance, but have not clearly shown reliability in scenarios that exhibit characteristics of changing activity encountered in the real world. Though not initially proposed as an IDS by Vaarandi, it is unknown whether SLCT or any other clustering-based IDS can achieve good performance without relying on user parameters or the content of the data being tested.

Regardless of whether SLCT can perform as an IDS by itself or not, its simple but powerful nature warrants further investigation into its possible uses. One such use may be the supervised creation of IDS signatures by utilizing the inherent summarizing nature of the algorithm. What is more, SLCT may be able to aid in the classificaton of new signatures in real time. Investigating SLCT can hopefully provide insight into the usefulness of data mining applications in the automation of cyber security.

In summary, the tasks for this work are as follows:

1. Test SLCT as a stand-alone IDS, by applying it in different ways. Try to determine the best possible user parameter configuration.

2. Test SLCT as a signature-creation algorithm using labeled ground truth.

3. Determine SLCT's ability to classify unknown signatures in a dynamic data set.

Chapter 2

Design and Implementation

This chapter discusses the design and implementation of the proposed work. The system consists of three components: Clustering-based IDS, Signature Creation and Signature Analysis, and Adaptive Signatures. The first component is designed to investigate SLCT's usefulness as a stand-alone IDS. The second component assesses the clustering algorithm's ability to create signatures from labeled ground truth. The third component addresses clustering used as a summarizing agent to help classify new kinds of incoming data. Section 2.4 discusses the implementation of the system. Throughout this chapter, data will refer to any log file, and clustering will refer to the algorithm proposed by Vaarandi, named SLCT.

2.1 Clustering-based IDS

This section will first describe the algorithm originally proposed by Risto Vaarandi as a log-analysis tool named SLCT. Its adoption as a stand-alone IDS will then be discussed.

2.1.1 The SLCT Algorithm

Let a data point be a single log file entry and its attributes are the words contained in it. The following definitions are made. The data space has a dimensionality of n, where n is the maximum number of attributes in a data point, where each attribute belongs to a field, f_i. We call Λ the set of all possible fields in the data, or, $\Lambda = \{f_1, ..., f_n\}$. A region S is

a subset of the data space, where certain attributes i_1,...,i_k ($1 \leq k \leq n$) of all points that belong to S have identical values v_1,...,v_k:

$$\forall x \in S, x_{i1} = v_1, ..., x_{ik} = v_k \tag{2.1}$$

We call the set (v_1,i_1),...,(v_k,i_k) the set of fixed attributes of region S. If k=1 (i.e., there is just one fixed attribute), the region is called a 1-region. A dense region is a region that contains at least N points (defined as a percentage of the total log lines), where N is the support threshold value given by the user [46]. A region, *i.e.*, a set of fixed attributes, is referred to as a cluster candidate if it contains at least one dense 1-region.

In preliminary experiments, it was determined that not all fields in Λ are necessarily useful for SLCT's operation. Therefore, we call Λ' the subset of Λ that was selected by the user that defines the fields which attributes may be used to define a region S. For reference, Table 2.1 summarizes the terms used in clustering throughout this thesis.

The SLCT algorithm consists of three steps: building a data summary, defining cluster candidates, and refining clusters. The first step, the pseudo-code for which can be found in Figure 2.1, identifies dense 1-regions, which are essentially words that occur in at least N percent of lines (the position in the log file line is also considered). A hash table or map can be used to efficiently store these values in [position, value] form. The number of times a [word, position] pair is found in a log line is known as the support value.

The next step, found in Figure 2.2 which is concerned with defining cluster candidates, iterates over each line checking for dense 1-regions. If a data point contains one or more of the identified frequent words, a cluster candidate is formed or the existing cluster candidate's support value is incremented. Vaarandi [46] explains this best:

> ... if the line belongs to m dense 1-regions that have fixed attributes (v_1,i_1), ..., (v_m,i_m), then the cluster candidate is a region with the set of fixed attributes (v_1,i_1), ..., (v_m,i_m). For example, if the line is Connection from 192.168.1.1, and there exist a dense 1-region with the fixed attribute (1, 'Connection') and another dense 1-region with the fixed attribute ('from', 2), then a region with

Symbol	Term	Description
w	1-region	A single value or word contained in a log file line.
Φ	Cluster	A cluster candidate whose support is above the support threshold.
ϕ	Cluster candidate	One or more fixed attributes that describes one or more similar log lines.
L	Data point	A single line of information in a log file containing multiple fields.
W	Dense 1-region	A 1-region whose support is greater than the support threshold, N.
D	Dictionary	The list of 1-regions and their supports.
n	Dimensionality	The number of fields in a log file.
f_k	Field	A type of information in a data point with a specific position (e.g. 'protocol').
Λ'	Field Selection	The set of fields which are considered when clustering.
(v_k,i_k)	Fixed attribute	A [value, position] pair, many of which make a cluster definition.
σ	Set of Cluster candidates	The set of all current cluster candidates.
N	Support threshold	The user parameter which specifies a percentage of the total log lines.
s_w, s_c	Support values	The number of data points that contain the 1-region or belong to the cluster candidate.

Table 2.1: Description of Clustering Terms

```
WHILE (L ≠ NULL)
  FOR (a ∈ L.attributes)
    IF(a ∈ Λ' )
      IF(a ∈ D )
        D.UpdateAttributeSupport(a)
      ELSE
        D.Add(a)
      END IF
    END IF
  END IF
  NUM_LINES++
    L = NextLine()
END WHILE
```

Figure 2.1: SLCT Stage 1: build Dictionary (D)

the set of fixed attributes ('Connection', 1), ('from', 2) becomes the cluster candidate [46].

In this way, each log line is assigned to exactly one cluster candidate. Let σ represent the set of these candidates.

```
ResetInput()
L = NextLine()
WHILE (L ≠ NULL)
   φ = new cluster
   FOR(a ∈ L.attributes)
     IF(a.support/NUM_LINES ≥ N)
       φ.attributes.Add(a)
     END IF
   END FOR
   IF (φ.attributes.Count > 0)
     φ.members.Add(L)
     IF (φ ∈ σ)
       φ.support++
     ELSE
       σ.Add(φ)
     END IF
   ELSE
     Anomalies.Add(L)
   END IF
   L = NextLine()
END WHILE
```

Figure 2.2: SLCT Stage 2: build set of cluster candidates (σ)

The third step, found in Figure 2.3 which refines cluster candidates into valid clusters, involves simply iterating over the cluster candidates to check their support value. If the support value, as a percent of total activity, is less than the user-defined support threshold N, then it is not considered a cluster. Vaarandi also proposes an iterative solution to fine tuning the support threshold. Clustering results may heavily depend on the threshold N, and therefore will be chosen based on extensive experimentation.

```
FOR (φ ∈ σ)
   IF (φ.support/NUM_L INES ≥ N)
      φ.valid = true (φ → Φ)
      FOR (L ∈ Φ.members)
         AssignClusteredStatus(L)
      END FOR
      FOR (L ∈ Anomalies)
         AssignAnomalyStatus(L)
      END FOR
   END IF
END FOR
```

Figure 2.3: SLCT Stage 3: select valid clusters

2.1.2 Application of SLCT: Legitimate Clusters

Intuitively, cyber attacks are assumed to be anomalous. This assumption is made by many related works mentioned above, and leads to the conclusion that any activity belonging to valid clusters created by SLCT must be normal activity. Any data that does not belong to any cluster must therefore be malicious activity. The function $AssignClusteredStatus(L)$ therefore sets L's status to normal, and $AssignAnomalyStatus(L)$ sets L's status to malicious. This method of using SLCT to classify unknown data by labeling members of valid clusters as normal activity we will refer to as SLCT-norm.

2.1.3 Application of SLCT: Malicious Clusters

Alternatively, regarding clusters created by SLCT as malicious activity may at first seem counter-intuitive, but there are many reasons why this method is justified. Normal activity is often harder to describe than malicious activity. One potential reason for this is that there are significantly more types of normal activity performed by an average host or network than there are types of known malicious activities that are effective. Cyber attacks are also often automated, making the activity produced very predictable. This automation, which can make an attack more effective and more destructive, can therefore be used to our

advantage. Indeed, a signature-based IDS system is built on the concept of detecting the recognizable pattern of malicious activity. If SLCT is concerned with grouping together similar data, then it follows that malicious activity will most certainly form a cluster if the activity forms a significant enough part of the observed data.

One clear problem with this method is that not only malicious but also normal activity will form clusters. However, if we assume that a cluster contains either mostly malicious or mostly normal activity, the problem then simplifies to differentiating between clusters. This assumption will later be verified, justifying our simplification of the problem. One characteristic of a cluster is that the number of fixed attributes in a cluster describes how similar the data in the cluster is: a higher number of fixed attributes indicates a higher degree of similarity. Given the automated, and therefore patterned nature of malicious activity, we assert that clusters formed from malicious activity often contain more fixed attributes than clusters formed from normal activity. If this is the case, then separating malicious clusters from normal clusters becomes a problem of separating clusters with more fixed attributes from the ones with less.

For this purpose, a new user parameter, M, is introduced that specifies the percent of fixed attributes out of the maximum possible that a valid cluster is required to have. An M value of 0 will allow a cluster to be formed regardless of the number of fixed attributes. Setting M to a higher value will ideally eliminate all normal clusters leaving only the malicious ones, and thus classifying the original data. We will refer to this method using the M parameter as a similarity filter on a mixed group of clusters as SLCT_attack. Figure 2.4 shows the final stage of SLCT modified to use the M parameter, with the modification shown in bold. The only differences between SLCT_norm and SLCT_attack are this modification, as well as the functions $AssignClusteredStatus(L)$ and $AssignAnomalyStatus(L)$. For SLCT_attack, $AssignClusteredStatus(L)$ sets each cluster member to malicious, and $AssignAnomalyStatus(L)$ sets each non-clustered line to normal. For SLCT_norm, it is the opposite.

Using SLCT in these two different methods should provide insight into the usefulness

```
FOR (φ ∈ σ)
   IF (φ.support > N AND φ.attributes.count ≥ M)
      φ.valid = true (φ → Φ)
      FOR (L ∈ Φ.members)
         AssignClusteredStatus(L)
      END FOR
      FOR (L ∈ Anomalies)
         AssignAnomalyStatus(L)
      END FOR
   END IF
END FOR
```

Figure 2.4: SLCT Stage 3 with M Parameter

of a simple density-based clustering algorithm as an Intrusion Detection System. Because a density-based clustering algorithm simply classifies data based on relative differences, it may also be useful for creating IDS signatures from a labeled data set. In other words, if the classification of data has already been performed, SLCT may be useful in summarizing the characteristics of both normal and malicious activity, instead of trying to distinguish between the two.

2.2 Signature Analysis and Signature Creation

Signature-based Intrusion Detection Systems make use of misuse models, or attack signatures instead of a relative approach. They rely on a set of signatures that attempts to describe every known cyber attack that could be carried out. One obvious flaw is that it is impossible to precisely describe new kinds of future exploits, otherwise countermeasures would already be in place. However, because a signature-based system can describe known attacks precisely, they often have very high detect rates. It is therefore essential to the system that the set of signatures is as comprehensive as possible given the set of historical data.

2.2.1 Siganture Analysis

For our purposes, a signature is similar to a cluster in that it contains a number of [value,position] attributes that describes activity. Activity matches a signature when all of the attributes in the signature match the corresponding values in the fields indicated by the positions in the signature. Other methods of matching signatures to activity can be investigated in future work. A signature conflict arises when there are two signatures that match the same activity with the same number of attributes. These attributes must be associated with different fields, otherwise the two signatures would be the same signature. There should be no conflicts between signatures created during the Signature Creation algorithm. For conflicts between historical signatures created during training and signatures classified by the New Signature Classification algorithm, discussed in Section 2.3, the historical signatures always take precedence.

2.2.2 Clustering for Signature Creation

To make the set of signatures as complete as possible, we propose employing SLCT to create clusters and therefore signatures from a set of labeled training data. A density-based clustering algorithm can be used as a tool which summarizes activity. The result of SLCT is essentially a set of groups representing the commonalities existing in the data, as well as identifying the data that is not sufficiently similar to any other data. Using this premise, we can apply SLCT to creating signatures that can later be used to identify malicious and normal activity.

In the IDS community, a labeled data set refers to data where each piece of data has a status associated with it: either a specific kind of attack or an occurrence of normal activity. If there are T attack types, let Ω be the set of all attack types $\{t_1, t_2, \ldots, t_T\}$, where t_i is the i^{th} type. Given a data set that contains a mix of malicious and normal activity that has been labeled we can seperate the data into T subsets corresponding to each type of attack. For each of these subsets, we can run SLCT to find the existing commonalities that exist

for that type of attack. The fixed attributes of the clusters produced from this method are then considered signatures. Signatures can be created from the set of normal lines as well.

2.2.3 Iterative Parameter Selection

One apparent problem with applying SLCT as a summarizing agent is the selection of the user parameter N to ensure that the highest amount of data is summarized. We assert that the more data successfully summarized, the higher the detection rate will be. An N value too high may exclude many lines from valid clusters, creating false negatives, whereas an N value too small will create too many clusters and thus defeat the purpose of summarizing the data. We address this by making the clustering process iterative based on the percent of data clustered, ρ. If the amount of data clustered is not above a threshold percent, the value of N is divided in half and SLCT is re-run on the data. It was found that a ρ threshold of 90% was sufficient to create signatures characteristic of the data represented. If, of course, the specified amount of data cannot be clustered without reducing the valid cluster support threshold to 1 member, an error is reported. In that case, the type of data could not be summarized by SLCT using the field selection chosen. Using this iterative process aims at producing a high detection rate while keeping the number of signatures to a minimum.

2.2.4 Signature Validation

Validation of these signatures is also important in obtaining reliable signatures. Given that the data is separated into training and testing data, we can exclude part of the training data from the signature creation process and instead use it for validating the signatures.

In our Signature Creation algorithm, validation will be used to reduce false positives by selecting different sets of fields for clustering. This approach is based on the premise that not all fields in the data are needed or necessarily useful in creating good signatures. A smaller number of fields will provide a more general summary of the data, allowing signatures to be effective despite small changes in the data. A larger number of fields will

yield a more specific description of the data, ideally reducing false positives, but becoming more resistant to variations in the same type of data. The validation process will start with the smallest, and therefore most general field selection. The field selection will get iteratively larger, and more specific, until the false positive rate is sufficiently low. The entire signature creation algorithm is illustrated in Figure 2.5.

```
FOR (t ∈ Ω)
   set_k = AllLinesOfType(t)
END FOR
set_normal = AllLinesOfType("normal")

FOR (t ∈ Ω)
   goCluster = true
   Λ' = NextFieldSelection()
   WHILE (goCluster)
      SLCT(set_k)
      IF (ρ ≥ 90%)
         Signatures = SLCT.validClusters
         results = SignatureAnalysis(validationSet, Signatures)
         IF (results.falsePos ≤ .00001 × validationSet.numLines)
            goCluster = false
         ELSE
            Λ' = NextFieldSelection()
         END IF
      ELSE
         N = N/2
      END IF
   END WHILE
   totalSignatures += Signatures
END FOR

SLCT(set_normal)
totalSignatures+ = SLCT.validClusters
results = SignatureAnalysis(validationSet, totalSignatures)
Report(attack − normal signature conflicts)
```

Figure 2.5: Signature Creation Algorithm

Once the entire Signature Creation algorithm finishes, the ideal results are at least one

signature for each kind of attack, plus at least one signature representing the normal activity contained in the data. Any conflict between a normal signature and attack signature is reported as an error. In this case, the algorithm could not create signatures for the attack type that can distinguish them from normal activity. The signatures can then be used to identify malicious and normal activity in the testing portion of the data. Though this process attempts to find the best signatures among a labeled data set, the signatures are static. Adapting these signatures to changes in the data is the more interesting part of this work.

2.3 Adaptive Signatures

Signatures used in a commercial IDS must be constantly updated to reflect changes in attacker methods. This process is often performed by an analyst, with the latency commonly on the order of days, or a week. This work investigates a method that attempts to incorporate attack information into the signatures immediately after the attack has occurred by using clustering.

Unlike some algorithms which attempt to use clustering as an anomaly-based sensor, we propose to use its summarizing abilities like those demonstrated in the Signature Creation Algorithm. In order to take advantage of density-based clustering, we utilize our previous assumption that clusters contain either mostly malicious or mostly normal activity. We also assume that there is a working set of signatures that must be updated to reflect a new type of activity present in the data.

First, attack and normal signatures that correspond to valid clusters are created from historical labeled data. These signatures define the known attack types, as well as a subset of the possible normal activity. New data enters the system in chunks, or windows. The size of this window can be important, which we will refer to as ψ. This could be achieved in a real system by periodically polling at a set rate in time. Each window is first clustered. The supports for new and existing words and cluster candidates are updated to form new or modified clusters. These clusters become the new set of signatures, which are then used to

analyze the same window. A set of challenges arise when considering this process.

2.3.1 Real-time Adaptation

First, clustering is by nature a post-processing, or data-mining application. For our purposes, we must adapt SLCT to real-time use while retaining a low computational complexity. We achieve this by introducing smoothing constants, α and β, to both the words' support and cluster candidates' support, respectively. In a window, the new support for a word and cluster candidate become:

$$s_w = s_w\alpha + s'_w(1-\alpha) \tag{2.2}$$

$$s_c = s_c\beta + s'_c(1-\beta) \tag{2.3}$$

where s_w and s_c are the current supports, and s'_w and s'_c are the new supports calculated from the new window. In this way we can continuously cluster incoming cyber data without having to process the entire history of data, and we refer to this modification of the algorithm as SLCT_RT (real time).

In order to approximate a real-time flow of data, further modifications to the method that reads in data was incorporated to adjust for the amount of data produced by a single attack. We previously defined a window as representing the data collected in a set amount of time. Therefore, some types of attacks which produce a large amount of log lines could skew a large number of windows. For example, a DoS attack such as *neptune* is aimed at flooding the network with traffic in a short amount of time. If a window represents a set amount of time, and if the window size in number of lines was kept constant, many successive windows would contain only these instances of *neptune*, making it seem like the attack takes a lot longer in time than it actually does. This would not be a realistic approximation in keeping with the rest of the data, where some attacks occur completely within one window surrounded by normal activity. To correct for this, the code that reads in the data was modified so that the window size only applies to normal lines. This confines consecutive attack lines completely to one window. It could be argued that this method is

both realistic and unrealistic, depending on the interpretation of one window, and how this approximation relates to a real-world scenario. If the system polled every one second, for instance, this would be an unrealistic approximation because it is unlikely that an entire attack would occur within this time frame.

In addition, to emulate varying amounts of activity, the size of each window is randomly chosen which centers around the mean window size ψ, specified by the user. This method was adopted because a constant amount of normal activity is fairly unrealistic for most hosts and networks. Both randomly choosing a window size and making attacks atomic within one window make up the component we will refer to as the Real-Time Emulator.

Applying a varying window size to only normal lines attempts to describe the process of collecting data from a set amount of time. For instance, a window size of 100 normal lines might correspond to 1 minute. Though approximating real time by observing data in windows may not accurately emulate polling data at a set rate, modifications were made to achieve the best possible approximation using the data set available.

2.3.2 New Signature Classification

Another problem with real time signature creation and adaptation is the classification of new signatures. After a window of data has been clustered, the set of fixed attributes for each cluster makes up the set of signatures. Many of these signatures will already have a label from the training phase. Any signature that has not been previously discovered is classified in the following way. Each word, or 1-region, in the dictionary has what is called an *attack support*. This value corresponds to the amount of data out of the total whose signature has contained the particular word and has already been classified as an attack. For instance, if the data is:

Red, **blue**, green : attack

yellow, **blue**, white : normal

red, grey, red : attack

purple, **blue**, black: attack

then the attribute (Blue, 2) would have an attack support of 0.667 because two of three pieces of data that contained the attribute were labeled as attacks. A classification for an unknown signature can therefore be given by taking the average of the attack supports for the signature. If this average is over a specified *confidence*, then the signature is for a new attack, else, it is normal. The nominal value of this confidence is of course 0.5, though a higher value may be used to possibly reduce false positives. Given a cluster that is yet unclassified, this process can be found in Figure 2.6.

$newSignature = \Phi$
FOR ($a \in \Phi.attributes$)
$sum = sum + a.attackSupport$
END FOR

IF($sum/\Phi.attributes.Count \geq confidence$)
$newSignature.status = newattack$
ELSE $newSignature.status = normal$

$Signatures.Add(newSignature)$

Figure 2.6: Signature Classification Algorithm

This method of classifying new signatures is contingent upon the assertion that attributes in the data are indicative of the status of the data. In other words, the process relies on an attribute occurring in mostly malicious or mostly normal activity. For our purposes, we will use ground truth to train the attributes' attack supports, and maintain those values throughout real time execution. The update of those attack supports could be done by an analyst or automatically based on the new signatures created.

2.4 Implementation of Proposed Work

All software was written in Microsoft Visual C# .NET on the Microsoft Visual Studio platform. Visual C# was the langauge chosen because it offers a very easily built GUI while

maintaining an organized Object Oriented Programming environment. A framework was built which allowed for rapid development of different algorithms, inputs, and outputs with easily configurable user parameters for each. For instance, each algorithm of interest was implemented as a subclass of a generic Algorithm Manager superclass. Figure 2.7 below shows a simplified version of the system diagram that represents the essential components of the software system that addresses the tasks listed in Section 1.4.

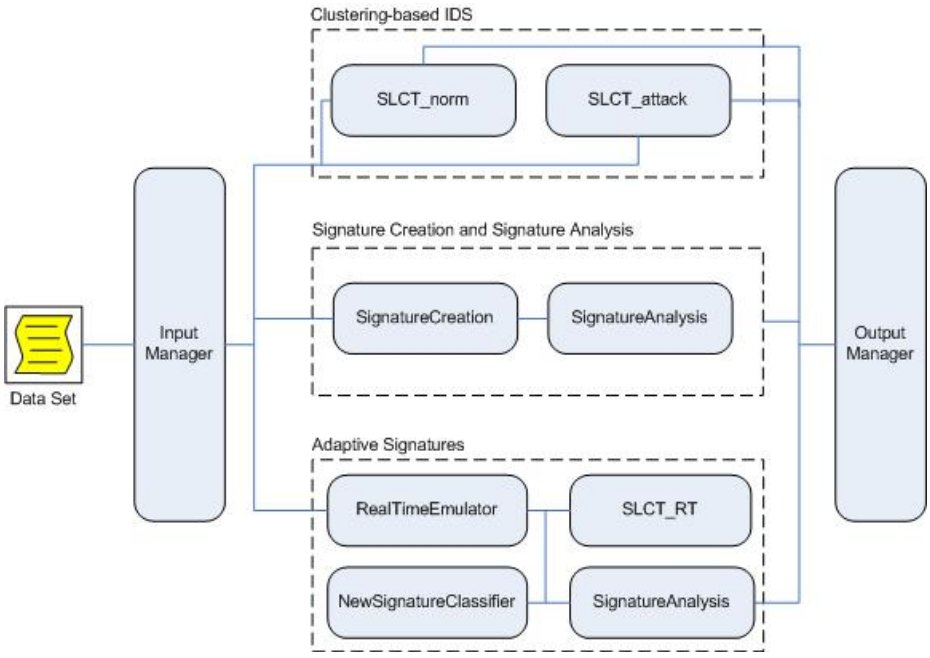

Figure 2.7: System Diagram

Chapter 3

Experiment Setup

3.1 Logs and Other Audit Data

On any computer network, there are many sources of information that could potentially be beneficial to an IDS. Some of this information is automatically logged, while others can be collected with specific tools. Though there is a vast amount of information, it is questionable as to how many sources would be of use in an IDS. These sources are usually divided into two categories: network-based and host-based.

Current IDS technology is usually focused on network-based sources, such as TCP traffic dumps. Although it is more or less accepted that network traffic contains enough information to identify malicious activity, one of the main reasons for this type of system is that it affords an easily scalable central IDS. However, network data does not constitute the whole picture, rather it isjust a reflection of what is actually occurring on one or more hosts.

Host data is slightly harder to make use of, due mostly to its large volume across a network. In fact, a single Windows server alone can contain the Application, Security, System, Directory Service, File Replication, and DNS Server logs, as well as the System Registry, System Call History and possibly Command History [45, 44, 43, 42]. The firewalls in a network can also provide their logs for analysis. Because of the volume and complexity of available host information, it is often unclear how it would be used in a useful and efficient IDS.

Though many types of network and host data can be collected in a real network, there are a few sets of data available that are recognized as being well suited for intrusion detection and prevention studies. Competitions, such as Defcon's Capture The Flag [8] and the KDD Cup [17], are often set up to test the abilities of public and private sector specialists. These data sets usually contain low-level data such as TCP dumps or some derivative thereof. These types of data sets are the best suited for our purposes, because it is at this low level that we are trying to adaptively detect intrusions.

3.1.1 The KDD Data Set

The 1999 DARPA Intrusion Detection Evaluation Data Set [22], produced by MIT's Lincoln Laboratories, includes disk dumps, network data, and audit logs from a simulated network where both legitimate and malicious activities were present. The network consisted of thousands of simulated hosts, with many common services available to both internal and external networks. However, ground truth for any of the available data sets is difficult to determine. Fortunately, analysts have processed the TCP data into data that contains labeled ground truth.

This processed data is the KDD Cup 1999 data set, formed for use in the Third International Knowledge Discovery and Data Mining Tools Competition [17]. The data set is a set of 'connection' records formed from MIT's DARPA sniffed network data. These connection records, numbering about five million, are labeled as either being normal or one of 21 different types of attacks from the following four categories: Denial of Service (DOS), Remote to Local (R2L), User to Root (U2R), and Serveillance / Probing. These attacks can be found in Table 3.1. The KDD data set does not contain time stamps or any temporal information besides the 'duration' field (index 0). The Real-time Emulator, described in Section 2.3.1, therefore attempts to approximate real time from these timeless 'connections'.

A DoS attack involves flooding a host or network with meaningless traffic, disrupting regular services. This type of attack can often resemble normal activity in the TCP header,

Category	Attack Type
Denial of Service (DoS)	back neptune ping of death (pod) smurf land teardrop
Remote to Local (R2L)	ftp_write warezclient warezmaster phf multihop spy imap guess_passwd
User to Root (U2R)	buffer_overflow loadmodule perl rootkit
Surveillance / probing	ipsweep nmap satan portsweep

Table 3.1: Attack Types

but the amount of traffic and other statistics can lead to an obvious detection. Remote to Local attacks involve a remote user trying to gain access to a host, such as by password guessing. This also may resemble normal activity, though it has distinguishable temporal differences from normal activity. User to Root attacks are privelage-escalation attacks that often take the form of some type of buffer overflow. Though these attacks started out being performed by manually modifying TCP packets, they can now be highly automated. Finally, surveillance attacks are exactly as the name implies: gaining information about a network and the services it provides. Port scanning is the most common method in this category, and can be detected by the obvious pattern of port increments.

The KDD data set contains 41 fields partitioned into three categories: TCP (0-8), content (9-21), and traffic data (22-40). Descriptions for these fields can be found in Tables 3.2, 3.3, and 3.4. Traffic data is computed over a 2 second window. Also, besides being conveniently labeled for use in ground truth comparison, the KDD data set is widely used in intrusion detection studies.

Field Name	**Description**	**Type**
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of "wrong" fragments	continuous
urgent	number of urgent packets	continuous

Table 3.2: TCP Fields Descriptions

3.2 Test Data Modifications

In order to properly test the proposed work described in Chapter 2, modifications were made to the KDD data. The modifications retain the characteristics of the data, such as the relative amounts of different types of attacks. The new data sets include those that contain

Field Name	Description	Type
hot	number of "hot" indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of "compromised" conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if "su root" command attempted; 0 otherwise	discrete
num_root	number of "root" accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_host_login	1 if the login belongs to the "hot" list; 0 otherwise	discrete
is_guest_login	1 if the login is a "guest"login; 0 otherwise	discrete

Table 3.3: Content Fields Descriptions

Field Name	Description	Type
count	number of connections to the same host as the current connection in the past two seconds	continuous
serror_rate	% of connections that have "SYN" errors	continuous
rerror_rate	% of connections that have "REJ" errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
srv_serror_rate	% of connections that have "SYN" errors	continuous
srv_rerror_rate	% of connections that have "REJ" errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous
The following fields apply to the destination host data		
dst_host_count	number of connections to the same host as the current connection in the past two seconds	continuous
dst_host_srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
dst_host_same_srv_rate	% of connections to the same service	continuous
dst_host_diff_srv_rate	% of connections to different services	continuous
dst_host_same_src_port_rate	% of connections that have the same source port	continuous
dst_host_srv_diff_host_rate	% of connections to different hosts	continuous
dst_host_serror_rate	% of connections that have "SYN" errors	continuous
dst_host_srv_serror_rate	% of connections that have "SYN" errors	continuous
dst_host_rerror_rate	% of connections that have "REJ" errors	continuous
dst_host_srv_rerror_rate	% of connections that have "REJ" errors	continuous

Table 3.4: Traffic Fields Descriptions

different amounts of malicious activity, those intended for training and testing, and data sets that attempt to emulate zero-day attacks.

A 10% subset of the KDD is also available which contains similar content as the original. Because the 10% subset requires much less resources than the full data set and is still representative of the content of the original, it will be used for all testing purposes. For the remainder of this thesis, any reference to the KDD data set will refer to the 10% subset just mentioned.

3.2.1 Malicious Content Variation

Many of the IDS systems and proposed algorithms discussed above gauge their performance on the KDD data set. None, however, present their results in terms of varying amounts of malicious activity. Very few mention the introduction of new kinds of attacks into the data. For Intrusion Detection Systems, these should be important characteristics of a proper test data set. To remedy this, the KDD data set was modified to reflect varying amounts and types of malicious activity. The first data set created from the KDD set is a modification of the amount of malicious activity contained in it. The original KDD data set contains approximately 80% malicious activity. To test our proposed work on a range of malicious volume, additional data sets were constructed with 0, 1, 5, 10, 25, and 50 percent attacks. This was done by iteratively removing every other attack line until the desired amount of malicious activity was reached. This preserved the relative amounts of different kinds of attacks existing in the original set, which is essential to testing algorithms that are concerned with the relative amounts of data in the set, such as density-based clustering. Table 3.5 shows a summary of the sets created.

3.2.2 Training, Validation, and Testing Sets

To test parts of the proposed work, such as signature creation, data sets for training, validation, and testing were needed. There are different methods to go about this, including

Subset Name	% Attacks	# lines
KDD_full	80	494021
KDD_50	50	194556
KDD_25	25	129703
KDD_10	10	108086
KDD_5	5	102397
KDD_1	1	98260
KDD_0	0	97278

Table 3.5: Malicious Content Modifications

k-fold Cross-Validation and Bootstrap [20, 10]. In Kohavi's analysis of these methods [20], he claims that k-fold Cross-Validation with a k value of 10-20 is superior for the data sets he used. Because the KDD data is similar to some of the data sets he used, such as the 'soybean large' data set with 35 fields and 19 label categories, Cross-Validation can be reliably used to partition the KDD data set. However, there are notable differences between the data sets in Kohavi's analysis and the KDD data.

The foremost of these differences is that the KDD set is much larger than the 'soybean' data set. This means that no 'cross' validation is needed because any selection will most likely contain an example of every type of data. The process in Figure 3.1 demonstrates the random selection and use of training, validation, and testing sets [7]. The size of the training set needed for the Signature Creation algorithm to properly function will be determined experimentally, rather than relying on the two-thirds shows in Figure 3.1.

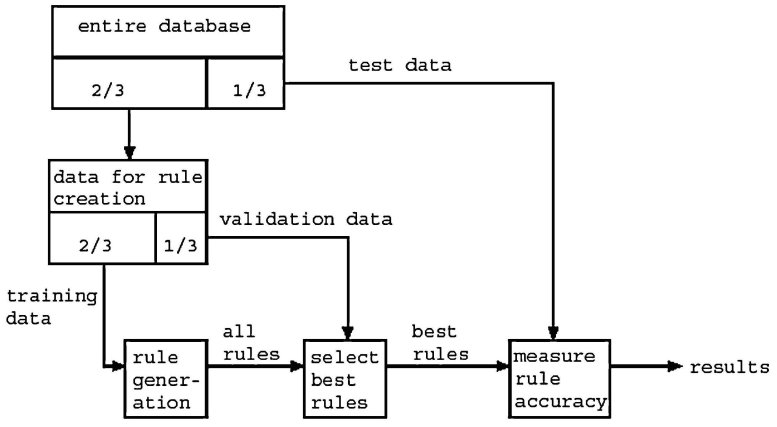

Figure 3.1: Training Set Partitioning

3.2.3 Dynamic Sets

The third set derived from the KDD set was an attempt to allow the system to approximate a real-time analysis of a changing data set. A data set was needed to introduce attacks that have not yet been processed by the system to determine its ability to adapt to new kinds of attacks. This was done in the following way. If there are T attack types, let Ω be the set of all attack types $\{t_1, t_2, \ldots, t_T\}$, where t_i is the i^{th} type. A subset of these attacks, Ω', is selected from Ω to represent the attacks that would 'enter' the data. Let Ω' be equal to $\{j_1, j_2, \ldots, j_L\}$, and L be the number of attack types in Ω'. Each instance of every attack type in Ω was removed from the original KDD data set, leaving only the normal lines. This new set with attack lines removed was divided into L+1 subsets. Let $\Delta = \{\delta_0, \delta_1, \ldots \delta_L\}$, where δ_i is the i^{th} of these subsets. Attack types from Ω' were added incrementally to each successive subset, starting with δ_1. Therefore, δ_1 contains only attack type j_1. The subset δ_2 contains attack types j_1 and j_2, and so on. The attack types found in Ω but not Ω' are added to δ_0, which is used for training SLCT and forming a baseline of signatures. Note that the relative amounts of the attacks found in the original set were preserved in each subset.

This process was repeated for different selections of Ω'. Recall from Table 3.1 that there are four categories of attacks. Each of these categories was designated seperately as Ω', as well as a selection of easily-clustered types according to the results of the experiment above in Section 3.5.2. We call the dynamic sets with different selections of Ω' KDD_dynamic_DoS, KDD_dynamic_R2L, KDD_dynamic_U2R, KDD_dynamic_surv, and KDD_dynamic_mix.

Malicious content, training and testing, and dynamic attack modifications to the KDD were used to test the three design components described in Chapter 2.

3.3 Performance Metrics

An essential part of performing an experiment is determining the measure of success and failure, and how the outcome can be used for other purposes. Table 3.6 defines the terms

used in IDS and clustering performance and characteristics. The two most common and useful metrics for intrusion detection are detection rate and false positive rate, defined below.

Symbol	Term	Description
A	Attacks	The number of lines that were attacks.
C	Clusters	The number of clusters.
a_i	Cluster Attack Fraction	The fraction of lines in cluster i that is malicious.
χ	Cluster Integrity	See Equation 3.3.
n_i	Cluster Normal Fraction	The fraction of lines in cluster i that is legitimate.
γ	False Positives	The number of lines that were incorrectly labeled as attacks.
ρ	Percent Clustered	The number of lines that belong to valid clusters out of the total number of lines.
π	Normals	The number of lines that were normal.
κ	Total Cluster Integrity	See Equation 3.4.
τ	True Positives	The number of attacks that were correctly labeled as such.

Table 3.6: Description of Performance Terms

$$DetectionRate(DR) = \frac{TruePositives}{Attacks} = \frac{\tau}{A} \tag{3.1}$$

$$FalsePositiveRate(FPR) = \frac{FalsePositives}{Normals} = \frac{\gamma}{\pi} \tag{3.2}$$

Receiver Operating Characteristic (ROC) curves may be generated to summarize the sensitivity of the system to user parameters. ROC curves are commonly used to show the performance of binary classifiers by showing detection rate versus false positive rate for different input parameter configurations. Though detection rate and false positive rate are the most commonly reported metrics for an IDS, other metrics may be used to provide insight into the system rather than reporting performance.

$$ClusterIntegrity = \chi = \frac{\sum_{i=0}^{C} |a_i - n_i|}{C} \tag{3.3}$$

$$TotalClusterIntegrity = \kappa = \frac{\sum_{i=0}^{C} |a_i - n_i|}{C} \times \rho \tag{3.4}$$

Using Cluster Integrity (χ) allows us to verify our previous assertion that clusters contain mostly malicious or mostly normal activity, and not a mix of the two. Total Cluster Integrity (κ) allows us to assess the clustering algorithm's ability to summarize the most amount of data while seperating malicious activity from normal activity.

3.4 Parameter Configurations

3.4.1 Threshold Values

There are a small number of user parameters for the algorithms discussed in Chapters 1 and 2. For SLCT, the two parameters are the support threshold N, and the field selection Λ'. The typical range of N can be anywhere from 0.001% to 10%, depending on the expected size of the data set. A small N value may specify a threshold of only 1 or less, which means every word will be considered dense and every line will be considered a cluster. An N value too high may not allow the creation of any dense words or clusters.

The M parameter, used in SLCT_attack has a range of 0% to 100%. An M value of 0% has no effect on SLCT, while a value of 100% enforces all fields in Λ' to be used for a valid cluster. Values in the range of 90% to 100% will be of interest, because it was noted in preliminary experiments that it is around those values that normal clusters are in fact distinguished from malicious ones.

3.4.2 Smoothing Constants

The real-time implementation of SLCT, named SLCT_RT, has two additional user parameters, α and β, which are smoothing constants for the supports of 1-regions and candidates.

These two values can have an effect on the sensitivity of the algorithm creating new clusters, as they determine how much historical support to consider as opposed to new support. The lower the value, the less historical support will determine the density of a 1-region or the validity of a cluster. With this in mind, a range of values from 0.1 to 0.9 will be investigated for α and β.

3.4.3 Window Size

The size of the window ψ that is used to approximate real time in the data sets is also important. The window size should be as small as possible while still being useful. A range of 100 normal lines to 10000 normal lines will provide insight into how small a window can be made before it cannot be usefully clustered by SLCT.

3.4.4 Field Selection

The field selection Λ' can take on any subset of the 41 fields in the KDD data set. There are, however, intuitive and interesting selections available that will be used. Indeed, it has been claimed by works such as [6] that the field selection is important in obtaining good classification results. The selections chosen can be found in Table 3.7. Some selections were chosen arbitrarily to determine if non-intuitive selections would be of any use. The three categories of fields (tcp, content, and traffic) were also chosen as seperate field selections to investigate their relative use. Refer to Tables 3.2, 3.3, and 3.4 for descriptions of the specific fields. Note that the names and indices of the fields apply only to the KDD Cup '99 data, which is described in Section 3.1.1.

A preliminary experiment was performed to determine the most useful fields to distinguish between malicious and normal lines. We define Field Integrity as how often specific attributes belonging to a particular field contribute to only normal or only malicious activity. A Field Integrity of 1 indicates that attributes in that field are always either a part of only normal activity or only a part of malicious activity. The Field Integrity for a field f_k

can be expressed in Equation 3.5 thus:

$$FieldIntegrity = \frac{\sum_{\forall(x_i,k)} |2 * (x_i, k).AttackSupport - 1|}{A} \tag{3.5}$$

where (x_0,k), ..., (x_A,k) are all the attributes that belong to field k. Using the expression $2 * AttackSupport - 1$ adjusts the attack support number to be highest when the field is most indicative of malicious or normal activity. An attribute's attack support of 0.5 means that the attribute is contained in an equal number of malicious and normal activity. It therefore is not a useful measure of how well a field can indicate malicious or normal status. We therefore assign a field integrity score of 0 to attributes with an attack support of 0.5. Likewise, we assign a field integrity score of 1 to attributes with attack supports of either 0 or 1. The final field integrity for field k is the average of these scores from individual attributes' attack supports. Using this definition, the Field Integrity was measured for each field in the subset of the KDD data containing 50% malicious activity to provide unbiased field integrity calculations. Figure 3.2 shows the results for each field.

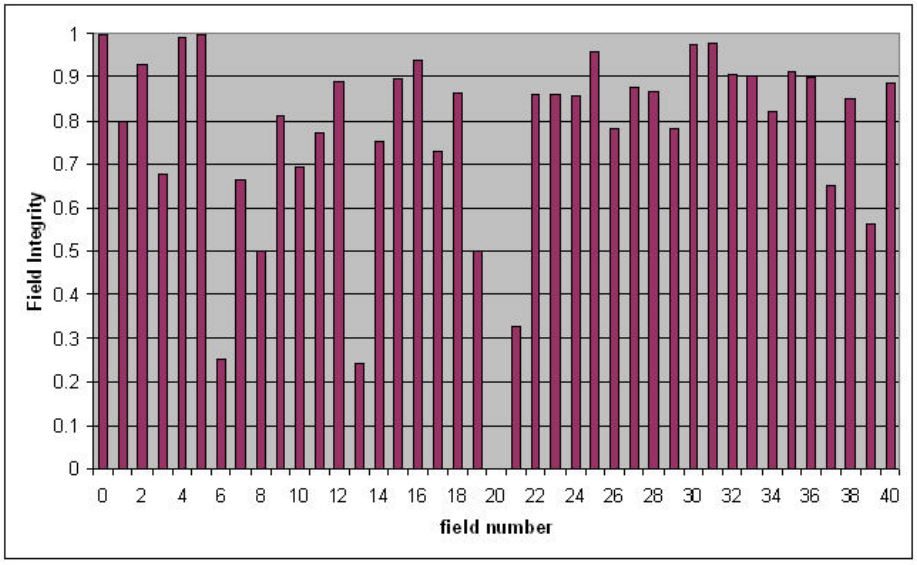

Figure 3.2: Measured Field Integrity

The fields with the highest Field Integrity should be the most indicative of a line's status.

These fields include 0, 2, 4, 5, 16, 25, 30, and 31.

Name	Field Numbers	Description
all	0,1,...,40	All fields in the KDD data.
first four tcp	0,1,2,3	The first four, which can provide much information about activity, contains duration, protocol, service, and the number of bytes from source to destination.
tcp	0,1,2,3,4,5,6,7,8	The TCP fields, see Table 3.2.
first half	0,1,...21	First half of the fields, arbitrary
second half	22,23,...40	Second half of the fields, which are also the Traffic fields. See Table 3.4.
second half odd	21,23...,39	Only odds of second half, arbitrary.
second half random	23,24,28,30,31,32,33,38,39	Very arbitrary selection.
middle	12,13,...,27	An arbitrary combination of Content fields and Traffic fields.
content	9,10,...,21	The Content fields, see Table 3.3.
experimental	0,2,4,5,16,25,30,31	Fields with the highest Field Integrity, found experimentally.

Table 3.7: Description of Λ' Selections

3.5 Experiment Procedure

Using the system proposed above as well as the modified versions of the KDD data set, a set of experiments were devised to investigate the different applications of clustering in intrusion detection.

3.5.1 Task 1: Clustering-based IDS

The first experiment involves determining the ability of clustering as an IDS itself. A ten percent subset of the KDD data was first modified for various amounts of malicious activity as described in Section 3.2.1. For both SLCT_norm and SLCT_attack, the effect of a range of N, M, and field selections on the detection rate and false positive rate was determined across the varying malicious activity subsets. In other words, the input parameter sensitivity was measured. For a general anomaly-based IDS, having a low parameter sensitivity is essential for reliable performance. For both SLCT_norm and SLCT_attack to be reliable

IDSs, they need a high detection rate, a low false positive rate, and an insensitivity to both input parameters as well as the amount and type of malicious activity present in the data set.

Another important investigation is on Cluster Integrity (χ) and Total Cluster Integrity (κ). A high χ will affirm our assertion that clusters contain either mostly malicious or mostly normal activity. A high κ will indicate the clustering algorithm's information summarizing ability, a characteristic which can in turn be used in signature formation and adaptation. As with detection rate and false positive rate, the effect of a range of input parameters on χ and κ should be determined to find the best combination of user parameters, N and Λ'.

3.5.2 Task 2: Signature Creation and Signature Analysis

The second experiment aims at finding acceptable conditions for signature creation using SLCT with regard to the size of the training and testing sets used. The Signature Creation algorithm is run on different sizes of training sets which produce sets of signatures. These signatures are then used to analyze the testing sets to find the detection rate and the false positive rate. Ultimately, this experiment will illustrate how well SLCT can create signatures under supervision. This experiment will also indicate how large a training set needs to be to create signatures, as well as the success of the signatures for each attack type.

3.5.3 Task 3: Adaptive Signatures

The third experiment aims at determining how well SLCT_RT can be used to adapt and create signatures in a dynamic situation. The first step in this process determines the best combination of user parameters, ψ, α, and β, that can maintain a high Total Integrity for SLCT_RT. A 33% training set out of the total experimented KDD data set is first clustered to establish a dictionary and set of cluster candidates. SLCT_RT is then run using the Real-Time Emulator on the remaining 66% testing set with varying user parameters to find the

best average Total Integrity for all windows.

Next, we must verify that SLCT_RT maintains a high Total Integrity which corresponds to new attacks in a data set forming clusters. To do this, we use the dynamic sets described above in Section 3.2.3. We will run SLCT_RT using the values found from the first step for ψ, α, and β on KDD_dynamic_DoS, KDD_dynamic_R2L, KDD_dynamic_U2R, and KDD_dynamic_surv. The first subset of the dynamic sets is used to train SLCT_RT. To save time, one attack from each data set will be singled out to be studied. The attacks chosen were *teardrop*, *guess_passwd*, *buffer_overflow*, and *portsweep* because there are a sufficient number of them, and provide a fairly difficult challenge to cluster. The percent of these attacks that successfully formed clusters will be reported for each dynamic set.

The final step adds the New Signature Classification algorithm. For each dynamic set, the first subset is used to create a base of signatures, as well as train SLCT_RT. The training phase is also used to collect attributes' attack supports using the ground truth of the training set. This process is only done once for each data set, though it could be a real time process based on the classifications given to new activity based on the set of signatures. Next, the dynamic testing data is analyzed. As new clusters are created in a window, they are classified according to the New Signature Classification algorithm, and the total set of signatures is then used to analyze the window. From this experiment, we should be able to determine the overall detection and false positive rates, as well as whether or not each new attack was succesfully clustered and classified.

Chapter 4

Simulation and Results

4.1 Clustering-based IDS

The clustering algorithm SLCT was tested as an IDS itself. The full range of user parameters was used to gauge the detection rate and false positive rate and determine the set of parameters that yield the best performance, which is ultimately gauged by detection rate and false positive rate. The algorithm's sensitivity describes how much these metrics deviate under different scenarios.

4.1.1 IDS Performance for SLCT_norm

SLCT_norm was used to classify each data set described in Section 3.5.1. In general, SLCT_norm exhibits a wide range of performance in terms of detect rate and false positive rate across the different data sets tested. First of, the algorithm was very reliant on the user parameters N and field selection. A ROC chart shown in Figure 4.1 illustrates this. Each point on the chart represents a different parameter configuration, where N is in the set $\{0.01, 0.1, 0.5, 1, 2, 5, 10\}$ and Λ' is any of the selections in Table 3.7.

The chart indicates that the algorithm is very sensitive to the user parameters because the points on the chart are very spread out. In essence, different user parameter selections will yield very different results. Selecting the correct user parameters for a given scenario would therefore be very critical.

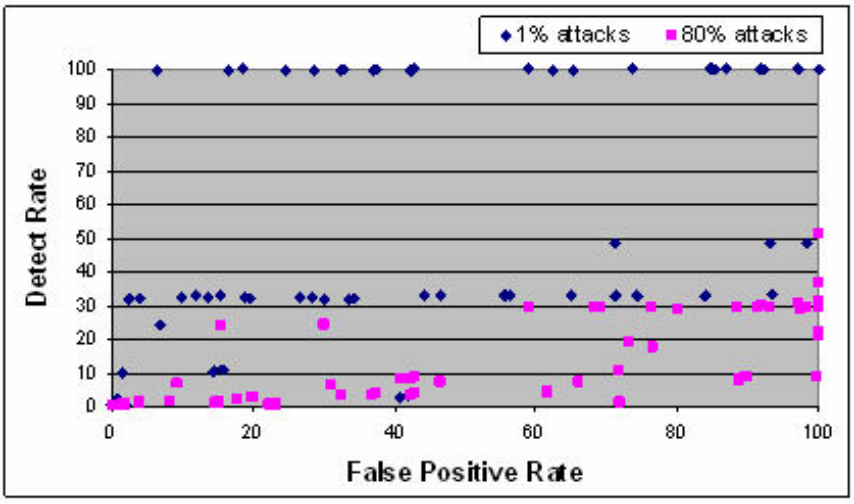

Figure 4.1: ROC Analysis for SLCT_norm

According to the ROC chart, SLCT_norm's performance is also very sensitive to the amount of malicious activity contained in the data set. For the same range of input parameters, the points for 1% malicious content are contained to a clearly different region that those for the 80% malicious content. Figure 4.2 shows this more clearly for one input parameter selection. We see that for a single selection of N and Λ', the detection rate will vary considerably across different amounts of malicious activity. This characteristic of SLCT_norm's detection rate is, of course, also related to the user parameter N. Despite the sensitivity to malicious content, Figure 4.2 also shows that cluster integrity is very high regardless of the relative content of the data, indicating that SLCT performs very well at seperating malicious and normal activity into seperate clusters. This will be illustrated further in Section 4.1.3.

For some parameter selections, SLCT_norm was able to achieve some insensitivity to malicious content. Figure 4.3 shows the performance across the sets which stays fairly even. The most obvious flaw, however, is that detection rate is very low and false positive rate is fairly high. The performance shown in Figure 4.3 was the best SLCT_norm could achieve in terms of insensitivity, not taking into account actual detection and false positive rates.

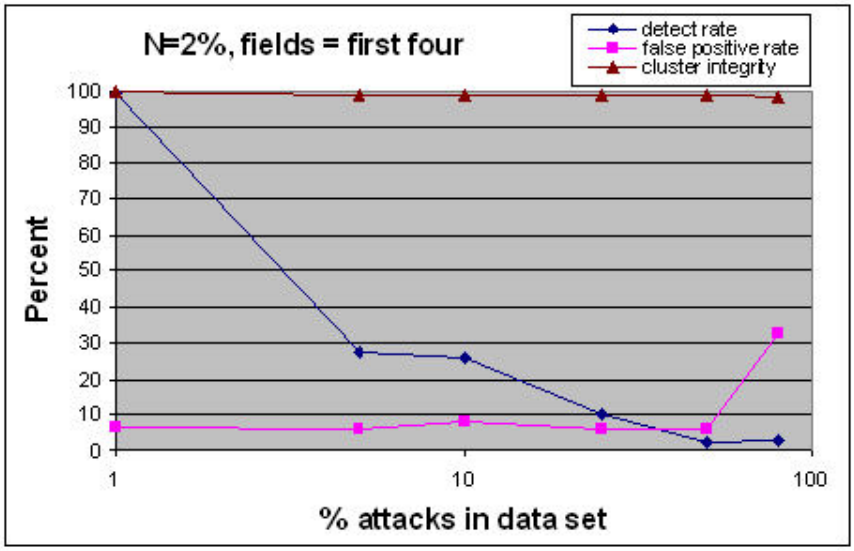

Figure 4.2: SLCT_norm Performance Across Varying Malicious Content

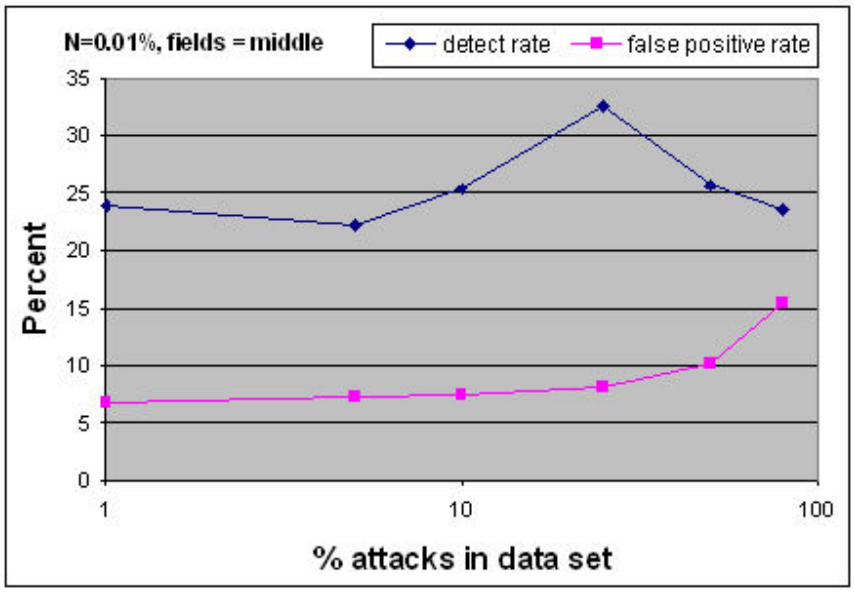

Figure 4.3: Acheived Insensitivity for SLCT_norm

4.1.2 IDS Performance for SLCT_attack

SLCT_norm did not perform well as a stand-alone IDS due to the dependence on user parameters as well as the amount of malicious activity in the data. SLCT_attack, the complementary approach, uses another user parameter, M, to try and eliminate this sensitivity by placing a threshold on the number of fixed attributes required for a valid cluster. Figure 4.4 shows one parameter configuration that achieved high detection rate and high false positive rate. Essentially, this corresponds to a high percent clustered, which was achieved by using a value of 0.1 for N. A larger Λ' selection was chosen so that more attributes would be available for the M threshold to take effect on. The 0% malicious content data set was also used so that the false positive rate during all normal activity could measured, which is an interesting result for unsupervised anomaly-based systems.

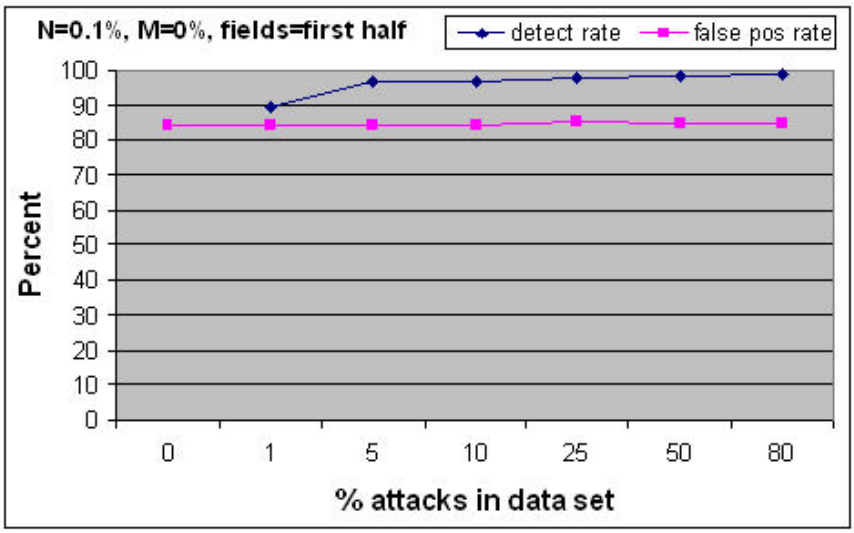

Figure 4.4: SLCT_attack Performance without M Parameter

In Chapter 2, we proposed that M may be able to help SLCT_attack differentiate between clusters that contain malicious activity and clusters that contain normal activity. Using a configuration that is able to cluster the most amount of data will allow us to demonstrate best the effect of the M parameter. Figure 4.5 shows the performance achieved when applying M. It is clear that the detection rate is high and the false positive rate is low

regardless of the malicious activity present in the data. This is the best configuration that could be achieved by SLCT as a clustering-based IDS. Considering the high detection rate and the relatively low false positive rate of current signature-based systems, however, this is not sufficient performance. The false positive rate is much too high to be of any real use, and the detect rate is probably not indicative of how many kinds of attacks were correctly labeled. However, Figure 4.5 does support our assertion that malicicous clusters often contain more attributes than normal ones because the M parameter was in fact able to distinguish between some malicious and normal clusters.

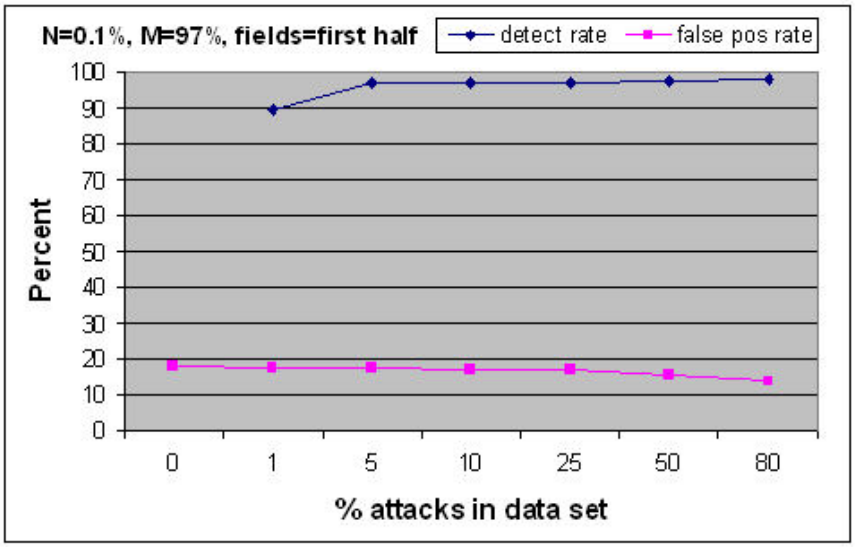

Figure 4.5: SLCT_attack Performance Using M Parameter

4.1.3 Cluster Integrity Performance

Though SLCT may not be well suited as a stand-alone IDS, it can possibly be used for other purposes. As briefly mentioned before and displayed in Figure 4.2, the Cluster Integrity (χ) was very high regardless of the amount of malicious activity present in the data. If χ is in fact a useful characteristic of SLCT's performance, then it follows that we should find the user parameter configuration that optimizes it. Figure 4.6 shows the Cluster Integrity on a data set with 50% malicious activity for a range of N and Λ'.

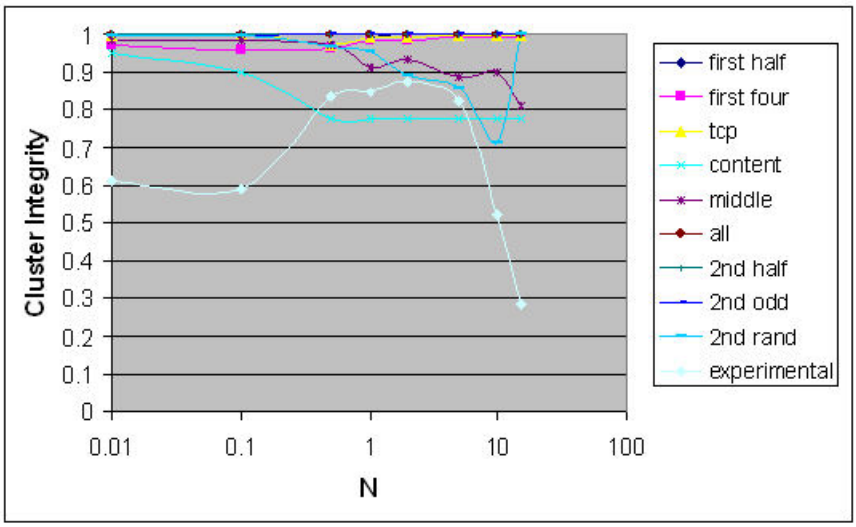

Figure 4.6: SLCT Cluster Integrity Sensitivity

Note that some field selections are slightly worse than others across a range of N, namely *content*, *middle*, and *second half random*. In any case, the high χ for most N and most fields verifies our previous assertion that clusters contain either mostly malicious or mostly normal activity.

Eventually, we are interested in SLCT's ability to summarize data. In other words, it must be able to cluster as much data as possible while still maintaining χ. This we defined as Total Integrity (κ). The first step is to determine which parameter configuration yields high ρ, percent clustered. Figure 4.7 shows these results. The critical parameter here is clearly field selection, and not N, yet the best value of N must still be determined. From Figure 4.7, we see that the Λ' selections that produce the highest ρ are *tcp*, *first four*, *first half*, and *content*. We can naturally eliminate *content* due to its relatively low cluster integrity seen in Figure 4.6.

Figure 4.8 shows the average Total Integrity and average number of clusters across all field selections for the specified values of N. Lower values of N, as expected, yield higher Total Integrity due mostly to a higher Percent Clustered. However, lower values of N also correspond to higher numbers of clusters as expected. Though a higher number of clusters will more accurately describe the data being clustered, it can become cumbersome for the

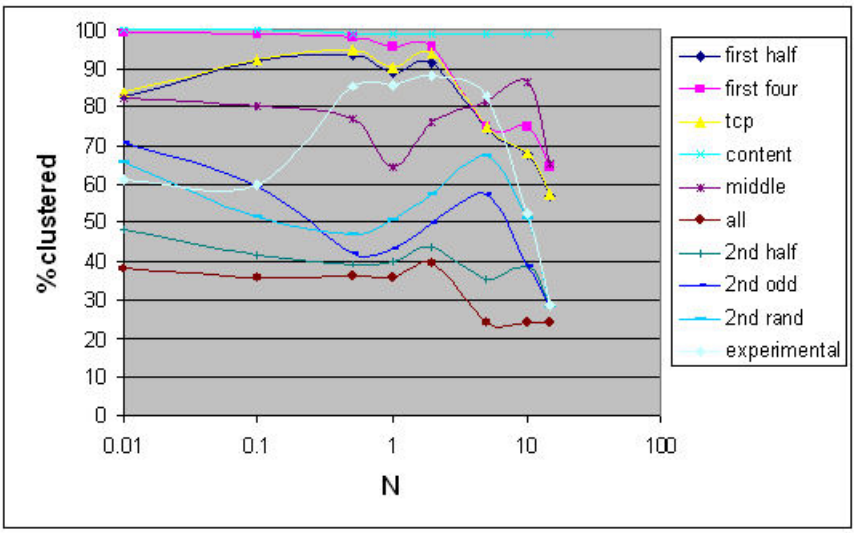

Figure 4.7: SLCT Percent Clustered Sensitivity

software in terms of computer resources. Since there is a local maximum of total integrity at $N = 2$ which yields a small number of clusters, this will be the value selected.

Now that the value of N has been decided, we can determine the field selection that yields the best κ. Figure 4.9 shows the Total Integrity for different field selections for $N = 2$. The error bars show one standard deviation of the Cluster Integrity for each cluster created, weighted by ρ. A high standard deviation would mean that some clusters had high Integrity, while others were much lower than the mean. It is clear that *first four* is the best selection having the highest average and a low standard deviation. The field selection *first four* is also convenient because it is a very small one, which is computationally very inexpensive.

4.1.4 Summary

The following conclusions were drawn concerning the usefulness of SLCT as an IDS:

- SLCT_norm does not perform well as an IDS because it is too sensitive to user parameters and the amount of malicious activity in the data. This is because clusters are too easily formed from malicious as well as normal activity.

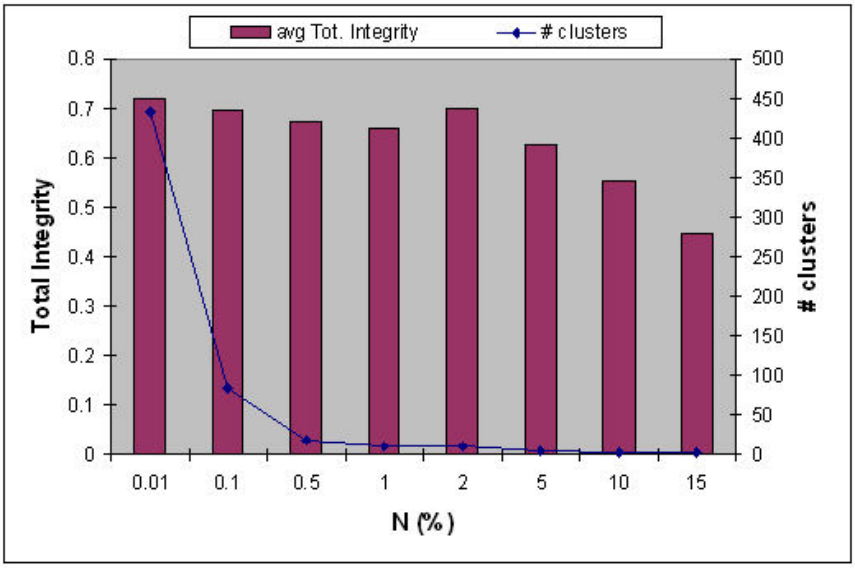

Figure 4.8: Average Total Integrity

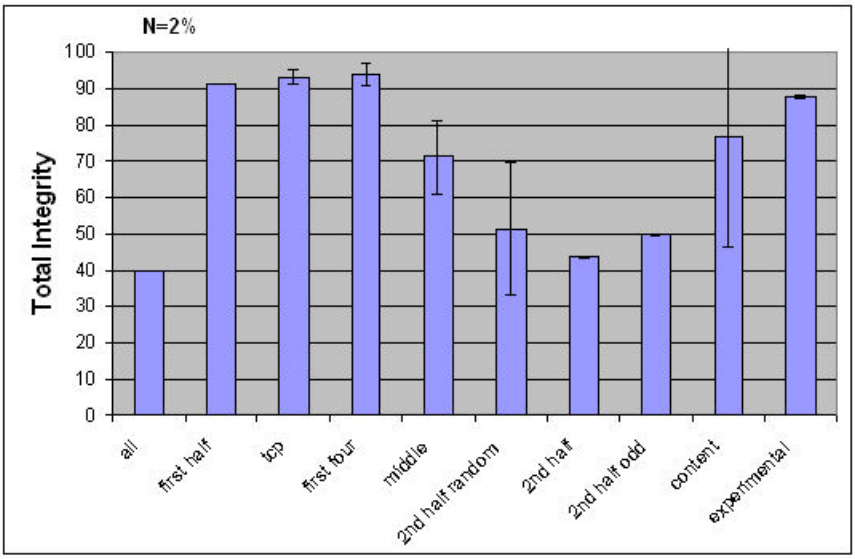

Figure 4.9: Total Cluster Integrity by Λ' Selection

- Using the M parameter in SLCT_attack can in fact improve SLCT's performance as an IDS, though it is still not satisfactory. Our assertion that malicious clusters contain more fixed attributes was verified.

- Cluster Integrity for SLCT is very high for some parameter configurations, verifying our assertion that valid clusters contain mostly malicious or mostly normal activity.

- Total Cluster Integrity is high for certain field selections. This can be taken advantage of when using SLCT's ability to summarize data.

- The best parameter configuration for a high Total Integrity is an N value less than or equal to 2, and a field selection of *first four*.

4.2 Signature Analysis and Signature Creation

The purpose of this section is to determine if SLCT can successfully create signatures from labeled training data. We can measure the success of this endeavor by looking at the detect rate and false positive rate of the attack signatures when the Signature Analysis method is run on the test data. Recall that the Signature Creation algorithm purposefully attempted to reduce false positives using the validation set, so these are not expected to be significant. Overall, the detection rate was around 99%, and the false positive rate much less than 1%. However, this is largely due to the flood of DoS-type attacks present in the data. For example, instances of '*neptune*' and '*smurf*' make up 95% of the malicious activity in the KDD set. Because of this large variance in the amounts of different attack types, it is more beneficial if we look at the performance for each attack type's signatures individually. Figure 4.10 shows the detect rates for each attack type for different sizes of training sets.

The attack types can be broken down into different categories depending on how their signatures performed:

1. Types which performed well regardless of training and testing set size.

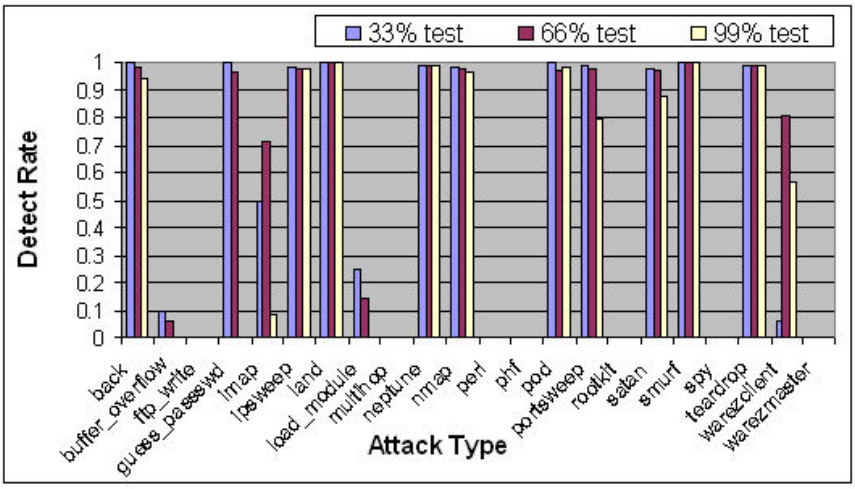

Figure 4.10: Signature Detect Rates

2. Types that had good signatures if enough lines were present in the training set.

3. Types that never created good signatures, possibly because little or no lines were present in the training set.

4. Types whose performance varied unpredictably with training set size.

Type 1, which generally performed well regardless of the training set size, were generally the DoS and surveillance attacks, including '*back*', '*ipsweep*', '*land*', '*neptune*', '*nmap*', '*pod*', '*portsweep*', '*satan*', '*smurf*', and '*teardrop*'. This category is by definition more predictable in nature. A DoS attack is a flood of the same activity, and surveillance contains predictable characteristics because it is concerned only with determining network characteristics. Additionally, the amount of network activity for these types of attack is typically higher than others, which means even a smaller training set will contain a sufficient number of attacks with which to form signatures.

Type 2 is similar to Type 1 in that the attacks are of a predictable nature, but a smaller training set may not contain a sufficient number to form good signatures. These include '*buffer_overflow*', '*load_module*', and '*guess_passwd*'. In this case, the signatures may not actually have a very good detect rate, but it is worth to noting that the trend decreases as

training set size decreases. It could be agrued, therefore, that signatures with very good detect rates could be created from a training set with more instances of the attacks.

Type 3 includes '*ftp_write*', '*multihop*', '*perl*', '*phf*', '*rootkit*', '*spy*', and '*warezmaster*'. Some of the training sets contained no lines for some of the attacks, and therefore did not yield any signatures. In some cases, because of the small amounts of some of these attacks in the original data set, the relative amounts were not preserved in the random selection of the training subsets. This made it difficult to determine the effect of training set size on signature performance. Other types, such as '*warezmaster*', are made up of activity that is too heterogeneous to be properly summarized by SLCT, and therefore difficult to create signatures for. These attack types labeled as Type 3 were those whose performance was never above 0%, and no conclusion could be drawn as to how well signatures could be automatically created for them.

Type 4, which includes only '*warezclient*' and '*imap*', do not seem to exhibit a direct relationship between detect rate and training set size. This phenomenon is probably related to the iterative nature of the Signature Creation algorithm as well as the randomly selected training and validation sets. Because the algorithm iterates over different values of Λ' until a low enough false positive is reached, signatures could use completely different fields from one training set to another. This simple fact alone could create the unpredictable differences in detect rate we see for these attack types.

It is clear that the Signature Creation algorithm proposed can create signatures with some success, but needs modification. A process which is able to more finely select better signatures would be superior to the one implemented.

4.3 Adaptive Signatures

The goal of this section is to demonstrate the ability of SLCT to classify new signatures in a changing data set. First, the parameters ψ, α, and β must be determined. Next we will demonstrate that SLCT_RT does form clusters from new attacks. Finally, the performance

of the New Signature Classification algorithm will be demonstrated.

4.3.1 Determining Window Size, Alpha, and Beta

The best values for a high Total Integrity for the SLCT parameters N and Λ' were already determined in Section 4.1. The modified version of SLCT for real-time used, dubbed SLCT_RT, has three more user parameters ψ, α, and β. Here we will determine these values of maintaining a high Total Integrity in near-real time; that is, from window to window.

In a real system, it would be most desireable to collect data in the shortest intervals possible. This would allow a timely response to any occurring attack. In our system, this translates to the smallest window size possible. A smaller window size will better emulate a real system; though it must be large enough to contain meaningful data to be of use to SLCT_RT. Using $N = 2$ and $\Lambda' =$ *first four*, window sizes of 100, 1000, and 10000 normal lines were tested with different combinations of α and β. Recall that SLCT_RT was first trained on a 33% subset of the KDD data to build a Dictionary and form preliminary cluster candidates. SLCT_RT was then run on the remaining 66%. Figure 4.11 shows the Total Integrity averages for the different parameter configurations. It is clear that a ψ value of 100 is not only sufficient, but superior to 1000 and 10000. This may be a result of the N value chosen. Also recall that a lower value for α and β puts more emphasis on recent activity than on historical data. As expected, a lower value for both α and β of 0.2 seems to work better than 0.9 in maintaining Total Integrity, as it allows SLCT_RT to adapt to the new activity instantly.

4.3.2 Clustering New Attacks

Now that the user parameters ψ, α, and β have been determined, it is time to test SLCT_RT on a data set where new attacks enter. Table 4.1 shows the final selection of user parameters.

Ω' describes the set of attacks that enter the data as emulated zero-day attacks, where the different selections were split into the four categories of attacks: DoS, R2L, U2R, and

Figure 4.11: Average Total Integrity

Parameter	Value
N	2
Λ'	*first four*
α	0.2
β	0.2
ψ	100

Table 4.1: SLCT_RT User Parameter Values

surveillance. Recall that only one attack from each of these categories was selected to be of interest. Four experiments were performed, one for each of the dynamic data sets. For example, SLCT_RT was run on the data set KDD_dynamic_DoS, and the amount clustered for the attack '*teardrop*' was the only metric measured. This was also done for the other three attack type categories and their respective dynamic data sets. At this point, we are only interested in verifying that new attacks do form clusters. Table 4.2 shows the results from this experiment. The column titled Number of Lines shows the number of lines that were the specific type of attack. The column titled Number of Windows was the number of windows that contained the specific attack, thus describing how spread out the attack lines were throughout the data set. The table shows that most attacks do form clusters. This means that signatures will be formed from these clusters, ideally being classified correctly and detecting future attacks. Note that the DoS attack '*teardrop*' had 100% clustered. This

is expected because by definition, a DoS attack is made up of a large volume of the same activity, which makes it easily clustered.

Attack	Type	% Clustered	Number of Lines	Number of Windows
teardrop	DoS	100	163	5
guess_passwd	R2L	92.45	53	2
buffer_overflow	U2R	84.84	33	13
portsweep	surv.	92.58	782	13

Table 4.2: Percent of Attacks Clustered with SLCT_RT

4.3.3 Classifying New Clusters in Simulated Real Time

The final step is to apply the New Signature Classification algorithm to SLCT_RT to determine its effectiveness in detecting attacks that have never been analyzed by the system. Table 4.3 shows the average detection rate and false positive rate for each of the Ω' selections: DoS, R2L, U2R, surveillance, and the mix of all four. Recall that these Ω' selections were described in Section 3.2.3. To provide more detailed insight into the system's performance, Table 4.4 shows the detection rate of each attack type.

Data Set	Detect Rate	False Positive Rate
KDD_dynamic_DoS	29.65	0.47
KDD_dynamic_R2L	72.72	7.54
KDD_dynamic_U2R	71.43	8.16
KDD_dynamic_surv	80.82	5.97
KDD_dynamic_mix	39.35	7.28

Table 4.3: Average Performance for Final System

One feature to note is the difference between the DoS results and the rest of the results. Both detection rate and false positive rate are significantly lower for DoS than for the other Ω' selections. This is caused by the signatures created for the DoS activity being mislabeled as normal. One explanation for this, is that the signatures created for DoS attacks are too similar to normal activity in the first four fields to be classified as attack signatures. For example, the signature for '*smurf*' can be easily determined by looking at the raw KDD

Data Set	Attack Type	Detect Rate (%)
KDD_dynamic_DoS	back	0.00
	Neptune	94.67
	Ping of Death	0.00
	Smurf	0.00
	land	4.76
	teardrop	0.00
KDD_dynamic_R2L	ftp_write	25.00
	warezclient	73.00
	warezmaster	80.00
	phf	50.00
	multihop	57.10
	spy	100.0
	imap	75.00
	guess_passwd	96.20
KDD_dynamic_U2R	buffer_overflow	70.00
	loadmodule	66.67
	perl	90.00
	rootkit	90.00
KDD_dynamic_surv	ipsweep	98.80
	nmap	95.65
	satan	27.13
	portsweep	95.00
KDD_dynamic_mix	teardrop	0.00
	guess_passwd	88.68
	buffer_overflow	90.00
	portsweep	91.83

Table 4.4: Detect Rate by Attack Type

data. Using the [word, position] format for attributes, this signature can be written as follows.

(0, 0) (icmp, 1) (ecr_i, 2) (SF, 3)

We also know from raw data inspection that the attributes (0, 0) and (SF, 3) are common in both malicious and normal activity. The success of this signature's classification therefore depends on the relative amounts of malicious and normal activity in the training set: more malicious activity will give these attributes higher attack supports. Recall that all attack lines that did not belong to the Ω' selection in the dynamic data sets were placed in the training subset, δ_0. Therefore, we know that the data set KDD_dynamic_DoS contains much less malicious activity in its δ_0 set than the other dynamic sets. This is because there is much more DoS activity than the other types of attacks. We can therefore conclude that one possible cause for the low detection rate of DoS attacks is the low attack support in the training set for common attributes such as (0, 0) and (SF, 3). Conversely, the same principle can be applied to explain the high false positive rates in the other selections of Ω' due to the large amount of DoS activity in the training sets.

One possible remedy to this weakness in the system's performance would be to acquire a real data set made in a similar manner to the modifications we performed on the KDD data. In other words, using a data set produced in a real environment intended to be used in this study would eliminate any effects that our modifications had on the results.

Also, further study into the use of distance-based clustering algorithms may prove insight into possible performance improvements. These algorithms make better use of continuous-valued fields, making it possible to use a larger selection of Λ', which would possibly increase accuracy.

The results obtained were similar to those collected in related works. The work in [2] reported an average of 66.6% detection rate for unknown attacks, though on a much smaller scale than our own. The results we obtained can be considered fairly good. The classification has obvious flaws, but shows promise because a good number of attack types

were more or less successfully detected. The false positive rate for many of the attack categories was too high, though maybe more analysis could be performed to reduce this.

Though the New Signature Classification algorithm did have some success in detecting new attacks, it is apparent that performance is reliant on the content of the training set, δ_0. This means that attributes' attack supports in only the first four fields can not reliably classify all new signatures.

4.3.4 Summary

The proposed system was to some degree successful in detecting new attacks. Performance, including both detection rate and false positive rate, were found to be slightly dependent on the attacks in the training subset. This conclusion leads us to believe that attributes with an established history and attack support are not indicative of a new signature's status. In other words, frequently occuring attributes will occur over time in significant amounts of both malicious and normal activity.

Chapter 5

Conclusions and Future Work

5.1 Conclusion

This work's main contribution was to investigate the use of SLCT, a density-based clustering algorithm, as it can be applied to Intrusion Detection. It was first tested as a stand-alone IDS with little luck, leading us to believe that few clustering algorithms would be of real use by themselves in real scenarios. However, SLCT does have the ability to seperate malicious from normal activity into clusters, summarizing a fairly large high-dimensional data set. Using these characteristics, SLCT was first incorporated into a Signature Creation algorithm. This algorithm could automatically produce signatures from ground truth with varying success. Some attack types did not yield very good results, while others were easily condensed into a small number of signatures.

SLCT was then adapted for real time use by adding the effect of smoothing constants to words' and candidates' supports, as well as using a window in the Real-Time Emulator. It was demonstrated that, for a data set where new attacks enter into it, SLCT can at least form clusters from those new attacks. Further, SLCT was with some success able to properly label those clusters to immediately identify malicious activity. The main limitation of the system was that common attributes' attack supports were skewed by large amounts of malicious activity in the training set. This method of clustering and classifying new data may be of use with further research to the Signature-based IDS community for detecting zero-day attacks in the short term, until more conventional and accurate signature creation

methods can be applied.

This method may also be of use in other domains besides intrusion detection. Identifying and classifying a new group of data which may have similar but not identical characteristics to historical data could be applied to any real-time binary classifier or data collection and summarization method.

The concepts proposed in this work are entirely experimental. Rigorous experimentation and fine-tuning would be necessary for systems similar to the ones proposed to be implemented in real system. The idea of attempting to classify a new signature shows promise, which could be augmented with the items addressed in the next section.

5.2 Future Work

5.2.1 Other Clustering Techniques

Many clustering algorithms currently exist, some of which are intended for Intrusion Detection while others are only intended for log file analysis. Other clustering algorithms, particularly distance-based ones, may provide other insights into clustering's uses for signature creation and adaptation. This would be an easily performed extension of the work, replacing SLCT and it's definitions of clusters and signatures with another algorithm that used a cluster centroid-based approach.

5.2.2 Temporal and Other Relationships in Clustering

As mentioned in Section 4.3.4, attributes alone are not wholly indicative of a siganture's status. The relationships between attributes, however, may provide further insight into a signature's classification. Temporal relationships would exist in a real network due to automated components, both legitimate and malicious. Studies on forming temporal relationships between attributes and cluster candidates may also prove to be useful.

5.2.3 Real Data or Implementation

The modifications made to the KDD data set were intended to produce data more representative of real network data characteristics. Having a data set made for testing varying amounts of malicious activity and detecting zero-day attacks would provide more accurate insight into the benefits of clustering in intrusion detection.

The proposed systems are also candidate for a real-time implementation in a test network. The algorithms themselves are completely suited for this purpose. The difficult part of this type of study would be in the selection of the data format, setting up the test network, and running background legitimate activity while executing specific attacks. Because the KDD data set was processed from TCP data, it could also presumably be done in real-time to stay consistent with the 'connection' format which is familiar to the systems currently implemented. However, other formats could be investigated.

Bibliography

[1] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery*, 11(1):5–33, July 2005.

[2] I. Bojanic. On-line adaptive ids scheme for detecting unknown network attacks using hmm models, 2005.

[3] Kalle Burbeck and Simin Nadjm-Tehrani. Adwice - anomaly detection with real-time incremental clustering. Dept. of Computer and Information Sci., Linkopings Universitet, Linkoping, Sweden, 2004.

[4] Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over an evolving data stream with noise. In *Proceedings of Sixth SIAM International Conference on Data Mining*, pages 328–339, Bethesda, MD, United States, April 2006. Department of Computer Science and Engineering, Fudan University, Shanghai, China, Society for Industrial and Applied Mathematics, Philadelphia, PA 19104-2688, United States. Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[5] N. Carey, G. Mohay, and A. Clark. Attack signature matching and discovery in systems employing heterogeneous ids. pages 245 – 54, Las Vegas, NV, USA, 2003. intrusion signature specification;multiple heterogeneous intrusion detection system;attack signature matching;multistage attacks;administrator interface;.

[6] S. Chavan, K. Shah, N. Dave, S. Mukherjee, A. Abraham, and S. Sanyal. Adaptive neuro-fuzzy intrusion detection systems. In *Proceedings. ITCC 2004. International Conference on Information Technology: Coding and Computing*, volume 1, pages 70–4, Las Vegas, NV, USA, April 2004. Inst. of Technol. for Women, SNDT Univ., Mumbai, India, IEEE Comput. Soc.

[7] A. Clare. Machine learning and data mining for yeast functional genomics, phd thesis. Department of Computer Science University of Wales, Aberystwyth, 2003.

[8] DEFCON conference. Defcon capture the flag (ctf) contest.

[9] O. Depren, M. Topallar, E. Anarim, and M. K. Ciliz. An intelligent intrusion detection system (ids) for anomaly and misuse detection in computer networks. *Expert Systems with Applications*, 29(4):713–22, 2005.

[10] B. Efron and R. J. Tibshirani. An introduction to the bootstrap. New York, NY, 1993. Chapman and Hall.

[11] M. Ester, H. P Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, pages 226–31, Portland, OR, USA, August 1996. Inst. for Comput. Sci., Munchen Univ., Germany, AAAI Press.

[12] Yong Feng, Zhong-Fu Wu, Kai-Gui Wu, Zhong-Yang Xiong, and Ying Zhou. An unsupervised anomaly intrusion detection algorithm based on swarm intelligence. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, volume 7, pages 3965–9, Guangzhou, China, August 2005. Coll. of Comput. Sci. and Technol., Chongqing Univ., China, IEEE.

[13] K. K. Frederick. Network intrusion detection signatures, part one. *Security Focus*, (1524), December 2001. `http://www.securityfocus.com/infocus/1524`.

[14] Venkatesh Ganti, Johannes Gehrke, and Raghu Ramakrishnan. Cactus - clustering categorical data using summaries. In *"Knowledge Discovery and Data Mining"*, page 83.

[15] Y. Guan, A. A. Ghorbani, and N. Belacel. Y-means: a clustering method for intrusion detection. In *Proceedings of CCECE 2003 - Canadian Conference on Electrical and Computer Engineering Toward a Caring and Humane Technology*, volume 2, pages 1083–6, Montreal, Que., Canada, May 2003. Fac. of Comput. Sci., New Brunswick Univ., Fredericton, NB, Canada, IEEE. Also available on CD-ROM in PDF format.

[16] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, 1998.

[17] S. Hettich and S. D. Bay. The uci kdd archive. http://kdd.ics.uci.edu, 1999.

[18] SRI International. Nides (next-generation intrusion detection expert system). http://www.csl.sri.com/projects/nides/.

[19] ShengYi Jiang, Xiaoyu Song, Hui Wang, Jian-Jun Han, and Qing-Hua Li. A clustering-based method for unsupervised intrusion detections. *Pattern Recognition Letters*, 27(7):802–10, May 2006.

[20] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *"Proceedings of International Joint Conference on Artificial Intelligence"*, pages 1137–43, August 1995.

[21] V. Kumar, J. Srivastava, Z. Zhang, and et. al. Minds - minnesota intrusion detection system. http://www.cs.umn.edu/research/MINDS.

[22] MIT Lincoln Laboratory. 1999 darpa intrusion detection evaluation data set, 1999. http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html.

[23] T. Lane and C. E. Brodley. Temporal sequence learning and data reduction for anomaly detection. In *Proceedings of 5th ACM Conference on Computer and Communications Security*, pages 150–8, San Francisco, CA, USA, November 1998. Sch. of Electr. and Comput. Eng., Purdue Univ., West Lafayette, IN, USA, ACM.

[24] Wenke Lee and Salvatore Stolfo. Data mining approaches for intrusion detection. In *Proceedings of the 7th USENIX Security Symposium*, 2005.

[25] Zhuowei Li, Amitabha Das, and Jianying Zhou. Usaid: Unifying signature-based and anomaly-based intrusion detection. In *Proceedings of 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD 2005*, volume 3518 NAI, pages 702–712, Hanoi, Viet Nam, May 2005. School of Computer Engineering, Nanyang Technological University, Singapore, 639798, Singapore, Springer Verlag, Heidelberg, D-69121, Germany. Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[26] Yong Guo Liu, Xiao Feng Liao, Xue Ming Li, and Zhong Fu Wu. A tabu clustering algorithm for intrusion detection. *Intelligent Data Analysis*, 8(4):325–44, 2004.

[27] B. T. Luke. Classifier construction kit distance metrics. http://fconyx.ncifcrf.gov/$\sim lukeb/ClassCK/cck-dmet.html$.

[28] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[29] S. Mukkamala, G. Janoski, and A. Sung. Intrusion detection using neural networks and support vector machines. In *Proceedings of 2002 International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1702–7, Honolulu, HI, USA, May 2002. Dept. of Comput. Sci., New Mexico Inst. of Min. and Technol., Socorro, NM, USA, IEEE.

[30] H. Nagesh, S. Goil, and A. Choudhary. Mafia: Efficient and scalable subspace clustering for very large data sets, June 1999.

[31] Inc Enterasys Networks. Dragon intrusion detection system. http://www.enterasys.com/products/ids/DSHSS7/.

[32] Sang-Hyun Oh, Jin-Suk Kang, Yung-Cheol Byun, Gyung-Leen Park, and Sang-Yong Byun. Intrusion detection based on clustering a data stream. In *Proceedings of Third ACIS International Conference on Software Engineering Research, Management and Applications*, pages 220–7, Mount Pleasant, MI, USA, August 2005. Dept. of Comput. Sci., Yonsei Univ., South Korea, IEEE Comput. Soc.

[33] Sang Hyun Oh and Won Suk Lee. An anomaly intrusion detection method by clustering normal user behavior. *Computers and Security*, 22(7):596–612, 2003.

[34] S. Patton, W. Yurcik, and D. Doss. An achilles' heel in signature-based ids: Squealing false positives in snort, 2001.

[35] P. A. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *"Proc. 20th NIST-NCSC National Information Systems Security Conference"*, page 365.

[36] L. Portnoy, E. Eskin, and S. Stolfo. Intrusion detection with unlabeled data using clustering. In *Proceedings of 2001 ACM Workshop on Data Mining Applied to Security (DMSA)*, 2001.

[37] M. Salour and Xiao Su. Dynamic two-layer signature-based ids with unequal databases. pages 6 pp. –, Las Vegas, NV, USA, 2007. dynamic two-layer signature-based IDS;unequal databases;signature-based detection;intrusion detection systems;.

[38] Tenable Network Security. Thunder. `http://www.tenablesecurity.com/products/thunder.shtml`.

[39] Karlton Sequeira and Mohammed Zaki. Admit: anomaly-based data mining for intrusions. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 386–395. ACM Press, 2002.

[40] H. Shah, J. Undercoffer, and A. Joshi. Fuzzy clustering for intrusion detection. In *Proceedings of the 12th IEEE International Conference on Fuzzy Systems (Cat. No.03CH37442)*, volume 2, pages 1274–8, St Louis, MO, USA, May 2003. Dept. of Comput. Sci. and Electr. Eng., Maryland Univ., Baltimore, MD, USA, IEEE. Also available on CD-ROM in PDF format.

[41] Sourcefire. Snort: an open source network intrusion prevention and detection system. `http://www.snort.org`, 2007.

[42] 3Logic Online Support. Event logs. `http://logic.linux8.com/support/eventlog.html`.

[43] Microsoft Online Support. Description of the microsoft windows registry. `http://support.microsoft.com/kb/256986`, November 15, 2006.

[44] Microsoft Online Support. How to move event viewer log files to another location in windows 2000 and in windows server 2003. `http://support.microsoft.com/kb/315417`, October 30, 2006.

[45] Microsoft Online Support. How to view and manage event logs in event viewer in windows xp. `http://support.microsoft.com/kb/308427`, April 26, 2006.

[46] R. Vaarandi. A data clustering algorithm for mining patterns from event logs. In *Proceedings of the 3rd IEEE Workshop on IP Operations and Management (IPOM 2003)*, pages 119–26, Kansas City, MO, USA, October 2003. Dept. of Comput. Eng., Tallinn Tech. Univ., Estonia, IEEE. Also availble on CD-ROM in PDF format.

[47] Ill-Young Weon, Doo Heon Song, and Chang-Hoon Lee. Effective intrusion detection model through the combination of a signature-based intrusion detection system and a

machine learning-based intrusion detection system. *Journal of Information Science and Engineering*, 22(6):1447–1464, 2006. Compilation and indexing terms, Copyright 2006 Elsevier Inc. All rights reserved.

[48] Ji-Qing Xian, Feng-Hua Lang, and Xian-Lun Tang. A novel intrusion detection method based on clonal selection clustering algorithm. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3905–10, Guangzhou, China, August 2005. Fac. of Autom., Chongqing Univ. of Posts and Telecommun., China, IEEE.

[49] Nong Ye and Xiangyang Li. A scalable clustering technique for intrusion signature recognition. In *2001 IEEE Workshop on Information Assurance and Security*, pages 1–4, 2001.

[50] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. BIRCH: an efficient data clustering method for very large databases. In *Proceedings of 1996 ACM SIGMOD International Conference on Management of Data*, pages 103–114, Montreal, Canada, 1996.

[51] Jiu-Ling Zhao, Jiu-Fen Zhao, and Jian-Jun Li. Intrusion detection based on clustering genetic algorithm. In *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, volume 6, pages 3911–14, Guangzhou, China, August 2005. Second Artillery Eng. Inst., Xi'an, China, IEEE.

[52] Shi Zhong, T. M. Khoshgoftaar, and S. V. Nath. A clustering approach to wireless network intrusion detection. In *Proceedings of ICTAI 2005 17th IEEE International Conference on Tools with Artificial Intelligence*, page 7, Hong Kong, China, November 2006. Dept. of Comput. Sci. and Eng., Florida Atlantic Univ., Boca Raton, FL, USA, IEEE.